UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Control de aplicaciones y *wearables* en smartphones Android mediante agentes conversacionales

Antonio Vaquerizo Muñoz
Tutor: José Colás Pasamontes

Julio 2016

Control de aplicaciones y wearables en smartphones Android mediante agentes conversacionales

AUTOR: Antonio Vaquerizo Muñoz
TUTOR: José Colás Pasamontes

HTCLab

Dpto. de Tecnología Electrónica y de las Comunicaciones

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Julio de 2016

Resumen (castellano)

Este trabajo de fin de grado está dividido en 2 partes, cada una de ellas describe el diseño e implementación de una aplicación para dispositivos diferentes y complementarios:

1. **Aplicación para móvil**: Plantea una metodología distinta a la hora de realizar aplicaciones con interfaces de usuario avanzadas, basadas en asistentes virtuales con capacidad de reconocer y sintetizar voz. Basándonos en esta metodología hemos desarrollado una aplicación que incorpora esos elementos y permite la interacción con otras aplicaciones a las que aporta la funcionalidad mencionada. Debido a ello, la hemos denominado "aplicación *gateway*". Gracias a ella, no es necesario un gran conocimiento de las tecnologías implicadas (asistente virtual, reconocimiento de voz y síntesis de voz). Esta aplicación tiene las siguientes características:
    a. ***Conexión con servicios en la nube (servicio de asistente virtual, servicio de reconocimiento y síntesis de voz)***, donde se encuentra ubicada la inteligencia del asistente virtual. Este se utiliza para mantener un diálogo natural con el usuario, orientado a la obtención de información o a la navegación en la aplicación de terceros a las que se quiere incorporar dicha funcionalidad.
    b. ***Configuración flexible y dinámica***, mediante un fichero de texto que contiene los parámetros de configuración. Gracias a este fichero, tanto la apariencia de la aplicación como las funcionalidades de la misma (dominio semántico de la aplicación, es decir, la base de conocimiento del asistente virtual) son flexibles.
    c. ***Interacción con Aplicaciones,*** tanto las que forman parte del sistema operativo Android de Google, como otras que dispongan de una interfaz para esa interconexión.
    d. ***Capacidad para conectarse e intercambiar datos con dispositivos externos* del tipo *smartwatch***, mediante el uso de un bluetooth.

2. **Aplicación para *wearable* tipo *smartwatch***: Creación de una aplicación modelo basada en una nueva metodología para *smartwatches* a la hora de acceder y visualizar los datos. Hasta ahora siempre se puso el foco en el móvil mientras que el *smartwatch* realizaba las funciones de complemento, recibiendo notificaciones. Con esta nueva estructura planteada hemos conseguido que el uso de la propia aplicación empiece y finalice en el *smartwatch* adquiriendo el mismo un papel mucho más relevante que el obtenido hasta el momento.

Abstract (English)

This Bachelor Thesis is divided in 2 parts, each part describes the design and the implementation of an application for different and complementary devices:

1. **Mobile application**: Propose a different methodology for developing applications with user interfaces based in virtual agents with the ability of recognizing and synthetize voice. Based on this methodology we have developed an application that incorporates this elements and enable the interaction with other applications, providing them the functionality mentioned. Because of that, the application has been called "Gateway Application". With it, you do not need a great knowledge of the technologies involved (virtual assistant, speech recognition and speech synthesis). This application has the following features:

    a. **Connection with cloud service (virtual assistant service, recognition and voice synthesis service),** where is located the intelligence of the virtual assistant. This is used to maintain a natural dialogue with the user, aimed at obtaining information or at navigating in third-party applications, which is wanted to incorporate this functionality.
    b. **Dynamic and flexible configuration,** by means of a text file that contains configuration parameters. Due to this file, the appearance and the functionality (semantic domain of the application, in other words, the knowledge root of the virtual assistant) of the application are flexible.
    c. **Interaction with applications**, both, applications with connection interface and applications part of Google android operating system.
    d. Ability to connect and exchange data with external devices such *smartwatches*, by using bluetooth.

2. **Wearable (*smartwatch*) application:** Create a model application based on a new methodology for *smartwatches* focused on access and view data. Until now, the focus was always on the mobile, while the *smartwatch* performed the additional functionalities, receiving notifications. With this new structure we have got that the use of the application starts and ends in the *smartwatch*, getting one new role much more relevant that the previous one.

Palabras clave (castellano)

Asistente Virtual, Aplicaciones, Android, *Smartwatch*, API, Fragmentos, Gateway, AIML, La nube, Fichero de configuración, Comandos de voz, Iteración, Integración, Mapas, Reconocedor de voz.

Keywords (inglés)

Virtual Assistant, Applications, Android, *Smartwatch*, API, Fragments, Gateway, AIML, cloud, Configuration file, Voice commands, Iteration, Integration, Maps, Speech Recognizer.

Agradecimientos

Lo primero agradecer a mi tutor José Colás su paciencia, apoyo y dedicación a lo largo de todo el periodo de tiempo que me ha llevado realizar y escribir este trabajo, otorgándome todo tipo de recursos sin los cuales, no habría sido posible terminar este proyecto. También quisiera agradecer el apoyo de todo el equipo de Vocalia, que me han tratado como uno más y me han ayudado a comprender el funcionamiento de la nube que ellos disponen. En especial a Roberto Ronzalen en la parte de implementación y pruebas, a Alberto Calvo en la parte estética y a Alberto Palero en el desarrollo de reglas y funcionamiento del asistente virtual.

A mis padres Pedro y Fabiola, por su apoyo constante durante tantos años y por todo lo que han hecho por mí y a mis abuelos que ya no están entre nosotros.

Gracias a mis amigos de toda la vida y a los que he tenido la suerte de conocer a lo largo de estos años, con los que he compartido momentos y experiencias.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

Cada vez más empresas en el desarrollo de asistentes virtuales, sin embargo, estos están enfocados en la interacción con sus propias aplicaciones, (ej. Google Now, Siri). Este trabajo de fin de grado pretende:

1. Ser una guía a la hora de desarrollar aplicaciones para Android que integren tecnología de asistentes virtuales avanzados en dispositivos móviles (smartphones y tablets) y que a su vez permitan el control de aplicaciones y dispositivos hardware *wearables* e IoT (Internet of Things) mediante el uso de la voz. Hoy en día, los dispositivos móviles con OS Android o iOS (Apple) disponen de asistentes virtuales con voz (ej. Google Now, Siri) que permiten la interacción del usuario con aplicaciones del dispositivo e incluso búsqueda de información sin necesidad de utilizar teclados. Sin embargo, no existen estándares que faciliten a otros desarrolladores la tarea de crear e integrar su propia tecnología de interacción avanzada multimodal con estos dispositivos y con los periféricos de los mismos de un modo sencillo. Este trabajo pretende ahondar en la definición de estas APIs[1] que posibilitan a asistentes virtuales desarrollados por otros fabricantes (no Google, no Apple) el control, la navegación y el acceso a la información sin necesidad de grandes desarrollos ni conocimientos sobre los dispositivos ni sobre la tecnología de los asistentes virtuales.

2. Proponer una nueva estructura a la hora de realizar aplicaciones en *smartwatches*.
   Los *smartwatches*, pese a ser uno de los dispositivos más extendidos y que más tiempo llevan en el mercado no significa que estén asentados. El mundo de los *smartwatches* aún se encuentra en fase beta. Todas las grandes compañías de la industria tecnológica están volcándose con estos productos y no hacen más que tantear y explorar diversas posibilidades. Como parte de esta experimentación, Samsung presentó el Samsung Gear S, un nuevo *smartwatch* con Tizen cuya principal característica es la inclusión de conectividad 3G, algo que ya se había visto en algunas statups pero nunca en una de las grandes del sector. Y es que hasta hace poco todas las compañías estaban asumiendo que los *wearables* deben funcionar de forma conjunta con el smartphone, sin embargo, desde la apuesta que hizo Samsung cada vez son más las compañías que se hacen la pregunta, ¿y si estamos siguiendo el camino equivocado? Con esta nueva estructura a la hora de realizar una aplicación en *smartwatches* se propone una usabilidad a caballo entre el reloj con conectividad 3G y el dispositivo completamente dependiente del smartphone, hasta el punto de que en numerosos casos no se pueda acceder a sus aplicaciones a no ser que llegue una notificación al mismo.

1.2 Objetivos

Los objetivos planteados en este TFG son:

[1] API: Application Programming Interface

- Crear una aplicación móvil que permita realizar de forma óptima la conexión con la nube donde se encuentra en nuestro caso la base de conocimiento del asistente virtual.

- Desarrollar un estándar que permita integrar en aplicaciones nuestro asistente y viceversa, para ello se hará un estudio de las diferentes formas posibles de desempeñar esta tarea y se llevará a cabo la que nos parezca óptima.

- Elaborar un pequeño estudio de los dispositivos *wearables* que hay actualmente en el mercado. Ver pros y contras de cada uno de ellos e introducir en el que se nos ajuste más a nuestros intereses, una app para poder realizar parte de la funcionalidad dispuesta en el terminal móvil.

- Desarrollar un estándar que permita una correcta comunicación entre móvil y *wearable*, para que de esta forma, se consiga tener 2 dispositivos maestros. Con ello, los dispositivos *wearable* que dependen de un smartphone para tener conectividad 3G tendrán algo más de libertad y con ella, los usuarios verán estos dispositivos más necesarios y útiles. Ya que actualmente sobre el 30 % de los compradores de *wearables* los devuelven debido a que los ven como elementos poco prácticos y no se acaban de acostumbrar al uso de los mismos.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Estado del arte.
- Estructura de la aplicación a realizar.
- Plataforma en la nube.
- Desarrollo de la aplicación para móvil.
- Desarrollo de la aplicación para *wearable*.
- Integración, pruebas y resultados.
- Conclusiones y trabajo futuro.

2 Estado del arte

2.1 ¿Qué es un *chatterbot*?

Según Pedro Jiménez Martín y Jesús Sánchez Allende [1], el término *chatterbot* fue acuñado en 1994 por Michael Mauldin con el fin de describir los programas de conversación. Este término surge como combinación de 2 palabras *Chatter*, término en ingles que significa parloteo y *Bot* que es el diminutivo de robot, (programa informático que imita el comportamiento de una persona).

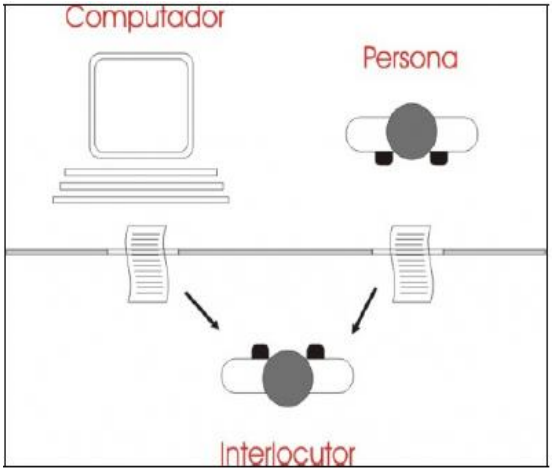


Figura 1: Esquema del test de Turing

Los *chatterbots* comenzaron como un elemento lúdico. Desde que en 1950 Alan Turing publicara "Computing machinery and Intelligence" previendo las preguntas que hoy están en el centro de la Inteligencia Artificial y es en este trabajo donde propone el Test de Turing, el cual sigue siendo la prueba a aplicar para responder a si una máquina es o no inteligente. El origen de esta prueba surge de la pregunta ¿Son capaces las máquinas de pensar? Ante esta cuestión Turing propuso verlo de otra forma más similar a un juego, al que llamo el juego de la imitación. El juego de la imitación se basaba en un escenario constituido por 3 individuos, 2 de los cuales eran humanos y el otro un ordenador. Uno de los individuos tenía que adivinar cuál de los otros dos era el humano solo mediante las respuestas escritas que reciba.

Algunos de los *chatterbots* clásicos más destacados que intentaron pasar el Test de Turing son los siguientes:

2.1.1 Eliza (1966)

Este programa fue diseñado entre 1964 y 1966 en el MIT (Instituto Tecnológico de Massachusetts) por Joseph Weizenbaum. La finalidad de este programa era evitar el hecho de que un programa estuviera basado en dominios limitados ligados a la estructura del propio programa. De esta forma cambiar de tema era algo tan sencillo como cambiar de modulo. Entre la variedad de módulos que disponía, el más famoso fue el que imitaba la metodología de un psicoanalista rogeriano (el psicoanalista Carl Rogers utilizaba una terapia que consistía en animar a los pacientes a hablar de sus problemas, respondiendo a sus preguntas con más preguntas). Este programa es actualmente un mito en la historia de la inteligencia artificial, y aunque a simple vista podría parecer que está entablando una conversación con una persona que le cuenta sus problemas, en realidad Eliza únicamente hace creer a su interlocutor que está hablando con un ser que razona y lo comprende. Sin embargo, esto lo consigue mediante una serie de trucos, como repetir frases cambiando yo por tu o utilizando una frase del principio

transformándola en otra pregunta relacionada con esa frase. A continuación se mostrará un ejemplo de conversación con Eliza [2]:

ELIZA: Please tell me what's been bothering you.

ME: Oh, hello, ELIZA. It's been a long time.

ELIZA: Hi. What seems to be your problem ?

ME: Lately, I've been feeling a little depressed.

ELIZA: And how do you feel right now ?

ME: I feel sad, but relieved to be writing.

ELIZA: Of what does feeling sad remind you ?

ME: Well, I'm an alcoholic, and I've been sober for a long time, but I drank yesterday. And then I decided to stop drinking again. But still; that makes me feel sad.

ELIZA: Is it because you are an alcoholic that you came to me ?

ME: I suppose that that's one of the reasons.

ELIZA: Can you elaborate on that ?

ME: I mean, it's not the only reason, but alcoholism is a primary thing that has messed up my life.

ELIZA: That's quite interesting.

2.1.2 Parry (1972)

Este *chatterbot*, como Eliza, también apareció en la década de los 70 y permitía mantener una conversación con relativa coherencia. Fue creado por el psiquiatra Kenneth Colby en 1972. Este programa implementaba el modelo de comportamiento de un esquizofrénico paranoico sobre la base de los conceptos, concepciones y creencias. Además, a diferencia de Eliza, incorpora una estrategia conversacional lo que le hacía más avanzado que Eliza, según algunos entendidos era como Eliza pero con actitud.

2.1.3 Jabberwacky (1982)

Este *chatterbot* fue creado por British cuyo programador es Rollo Carpentier y tenía como objetivo simular el comportamiento humano natural, en una charla amena, interesante y humorística. Fue el primer intento de crear inteligencia artificial a través de la interacción humana. Para ello, la tecnología que posee Jabberwacky es

diferente al otro software que se estaba desarrollando en aquel momento en la IA. Este sistema estaba diseñado para aprender el idioma y el contexto a través de la interacción con los seres humanos y por tanto, no hay reglas fijas ni principios programados en el sistema. El sistema almacena todas las conversaciones y comentarios de los usuarios e intenta utilizar esta información para encontrar la respuesta más adecuada a cada situación. Por esto, se trata del primer *chatterbot* con capacidad de aprender y responder de la forma adecuada si se le proporciona la suficiente información [3].

JABBERWACKY: Should I be talking to you?
ME: Theoretically, not.
JABBERWACKY: Why not?
ME: Because I will most likely cause you to question your morals and change your beliefs.
JABBERWACKY: Do you like me?
ME: What is it to like something? I cannot answer your question.
JABBERWACKY: Yes, ask something wiser, my wisdom is temporarily lowered.
ME: Perhaps you care to talk about philosophy?
JABBERWACKY: Perhaps you can figure it out yourself.
ME: Mayhaps. That would be rather difficult, I've been trying to figure it out every day.

2.1.4 Alice (1995)

ALICE (Artificial Lingüística Internet Computer Entity) es un proyecto de internet, que forma parte del Proyecto Pandora y que consiste en la creación de *chatterbots* de todo tipo, en especial los de *chat.* Desde la página web ALICE, el usuario puede entablar una conversación con un programa inteligente que simula una charla real, en la que el usuario puede difícilmente percatarse de que está hablando con un robot. Esta tecnología está desarrollada en Java por el Dr. Richards S. Wallace en inglés, encargado de la programación de *chatterbots* de Pandora, cuyo propósito no es otro que probar la capacidad de estos agentes inteligentes. Alice está inspirada en Eliza, y actualmente es uno de los *chatterbots* más fuertes de su tipo y ha ganado el LoebnerPrize, logrando la adjudicación de humanoide en tres ocasiones: 2000, 2001 y 2004. Además, cabe destacar que su mente está escrita en AIML un lenguaje basado en XML. Por otro lado la ingeniería de Alice y del AIML también son soportables en otros lenguajes de programación aparte de Java, como C, C++, PHP y muchos otros, lo que contribuyó a la popularidad de Alice y del, AIML. A causa de todos estos factores hoy en día hay más de 206000 *chatterbots* basados en la ingeniería de Alice y AIML.

2.2 Sistemas de diálogo en la actualidad

Todos estos *chatterbots* de los cuales hemos hablado han forjado los cimientos de los numerosos *chatterbots* que hay en la actualidad. Desde el proyecto Pandora y Alice han surgido numerosos *chatterbots* y no solo *chatterbots* sino una serie de sistemas de

diálogo y asistentes virtuales cada vez más eficientes, los cuales han dejado atrás, en gran medida, el lenguaje superfluo de los *chatterbots*, para dar lugar a una comunicación más enfocada en dar un servicio, este es el caso de la mayoría de sistemas de diálogo actuales. Este tipo de sistemas, se utilizan en gran medida en páginas web. Pero, ¿Cuándo conviene meterles en una página web?

Un asistente virtual es de vital importancia cuando:

- Se incrementa drásticamente el número de consultas por las vías más tradicionales como: llamadas, correo postal y correo electrónico.
- Las consultas son altamente repetitivas.
- Se desea devolver información homogénea a todos los usuarios o particularizar qué información se requiere proporcionar a cada tipo de usuario.
- Mejorar la imagen tecnológica de la compañía y atraer visitas a la página web, algo de vital importancia para el posicionamiento de la página en los buscadores.
- Se desea conocer que le interesa a cada cliente.

Si además el asistente virtual dispone de compresión de lenguaje natural con sistemas de diálogo esto producirá las siguientes ventajas añadidas:

- **Una mayor eficacia:** Permite el análisis sintáctico, morfológico y semántico de cualquier consulta y la interpretación de cada enunciado y de cada término introducido por el usuario, por lo que proporciona información más precisa que un buscador tradicional.

- **Mejora la satisfacción del cliente online** ya que está disponible 24 horas al día los 365 días del año en los ámbitos de atención al cliente. Y la configuración semántica mejora la capacidad de emular una conversación por lo que incrementa la satisfacción general del cliente.

- **Optimización de costes:** Según Forrester Research, contar con un asistente virtual que incorpore tecnología semántica permite reducir hasta 6 veces el coste de respuesta de un operador tradicional.

- **Aprendizaje automático continuado:** El aprendizaje progresivo a partir de la experiencia y fruto de las interacciones con los usuarios es una de las grandes fortalezas de los asistentes virtuales semánticos, que pueden incrementar con el tiempo su catálogo de respuestas.

- **Detecta tendencias y necesidades del mercado:** Al centralizar todas las vías de interacción con el mercado, las redes sociales, la web, el buscador y la atención online, desde un único punto, las compañías obtienen una visión más amplia del consumidor y, en consecuencia, de sus intereses y necesidades. Gracias a la tecnología semántica, es posible acumular y procesar datos de gran utilidad para la toma de decisiones estratégicas para el negocio.

Por otro lado, los asistentes virtuales en los últimos años han salido de las páginas web y se han introducido con gran impacto en los dispositivos móviles como *smartphones*, *tablets* o relojes inteligentes, y no solo eso, sino que también los mismos asistentes

virtuales que están presentes en estos dispositivos móviles están empezando a conquistar el novedoso mundo de IoT (Internet of Things), en el cual, según la página Artificial Solutions [5], existen 3 tipos de dispositivos:

1. Dispositivos personales principales, como los móviles inteligentes y la tecnología *wearable*.
2. Dispositivos compartidos en un ecosistema privado, como por ejemplo *smart TV*, plataformas de control de hogares digitales y coches inteligentes.
3. Dispositivos de dominio público tales como radares para el control de la velocidad.

De esta forma los dispositivos personales principales serán altamente personalizados e inteligentes y se usaran para controlar y gestionar los otros 2 tipos de dispositivos. Por lo tanto, se requerirán interfaces con reconocimiento de voz que sean capaces de mantener conversaciones en los distintos dispositivos, recordar conversaciones anteriores, comprender el contexto y ser capaces de razonar y reaccionar a las acciones que realice el usuario de forma inteligente. Por todo esto, la tendencia desde hace algunos años hasta la actualidad es dar una mayor importancia a los asistentes virtuales en los terminales móviles. Esto se debe a que son ellos los que al final desempeñarán el papel de controlar todos los distintos dispositivos que nos rodean en la vida cotidiana. De esta forma e intuyendo en gran medida la que iba a ser la tendencia tecnológica en los últimos años, las principales compañías tecnológicas de software dieron el salto a este mundo y apostaron por él. Así surgieron algunos de los asistentes virtuales más conocidos y usados de la actualidad: Siri, Google Now y Cortana, los cuales pertenecen a Apple, Google y Microsoft respectivamente.

2.2.1 Siri

Siri, es una aplicación con funciones de asistente personal para iOS, utiliza el procesamiento de lenguaje natural para responder, preguntar, hacer recomendaciones y realizar acciones mediante la delegación de solicitudes hacia un conjunto de servicios web. Siri fue creada en diciembre de 2007 por Dag Kittlaus (CEO), Adam Cheyer (VP Engineering) y Tom Gruber (VP Design) junto a Norman Winarsky del grupo SRI Venture Group. Tras problemas de viabilidad económica, Siri fue adquirida por Apple en 2010 y el 4 de octubre de 2011 se anunció que Siri se incluiría en el nuevo dispositivo móvil de Apple, el iPhone 4S. La nueva versión de Siri IOS7 está muy integrada y ofrece interacción conversacional con otras aplicaciones como los recordatorios, la consulta del estado del tiempo, la bolsa, el servicio de mensajería, el correo electrónico, el calendario, los contactos, las notas, la música, el reloj, el navegador web, mapas, etc. También es capaz de contestar más rápido a las preguntas más frecuentes y consultar fuentes como Bing, Wikipedia y Twitter, incluso se encarga de devolver llamadas, leer mensajes de buzón de voz, ajustar el brillo de la pantalla y muchas otras tareas.
Además Apple colabora con varios fabricantes de coches para integrar Siri en los sistemas de control por voz. Pulsando un botón del volante se puede llegar a preguntar a Siri sin apartar la vista de la carretera y para reducir aún más las distracciones, la pantalla del dispositivo iOS no se iluminará. La prestación *Eyes Free* permite usar Siri para hacer llamadas, seleccionar y reproducir canciones, escuchar y redactar mensajes de texto, usar Mapas y obtener indicaciones en ruta, leer las notificaciones, encontrar información del calendario, añadir recordatorios y muchas otras tareas más.

2.2.2 Google Now

Google Now es un asistente personal inteligente desarrollado por Google que está disponible dentro de la aplicación de Google Search para dispositivos móviles con sistemas operativos Android. Google Now usa una interfaz de usuario con lenguaje natural para hacer recomendaciones, hacer preguntas y realizar acciones mediante la delegación de las solicitudes a un conjunto de servicios web. Junto con la respuesta a las consultas iniciadas por el usuario, Google ofrece ahora de forma pasiva la información al usuario mediante predicciones, de esta forma se anticipa a lo que necesita en función de sus hábitos de búsqueda. El primer smartphone en el que estuvo disponible fue en el Galaxy Nexus desarrollado por Google y ensamblado por el fabricante Samsung. A diferencia de Siri, el servicio estuvo disponible para iOS desde el 29 de abril de 2013. Entre sus haberes en su corta trayectoria, el premio a la *Innovación del año en 2012* por la revista de ciencia y tecnología americana *Popular Science*. Google Now está implementado como un aspecto de la aplicación Google Search y es capaz de reconocer acciones repetidas que el usuario realiza en su dispositivo (ubicaciones comunes, citas repetidas del calendario, consultas de búsqueda, etc.) para mostrar la información más relevante al usuario en forma de tarjetas. El sistema aprovecha el proyecto Knowledge Graph de Google, que es un sistema usado para ensamblar los resultados de búsquedas más detallados mediante el análisis de su significado y las conexiones. Además Google Now ha introducido un soporte multilenguaje que es capaz de conocer hasta 7 lenguajes al mismo tiempo, los cuales deberán ser seleccionados por el usuario de forma manual para que sean reconocidos. Otra novedad relevante, es la relacionada con el comando de voz "OK Google" con el cual podremos utilizar comandos de voz sin necesidad de usar el teclado para empezar a reconocer comandos de voz.

2.2.3 Cortana

Cortana es el asistente virtual de Microsoft, fue presentado en abril de 2014. Según el investigador que contribuyo en el desarrollo de la inteligencia artificial en la que se basa Cortana, Larry Heck esta ha sido creada por Windows para que se haga más inteligente a medida que el usuario la use. Este compromiso de Microsoft con este tipo de enfoque permitirá al asistente virtual proporcionar una experiencia más parecida a interactuar con una persona real de la que ofrecen sus competidores como Siri o Google Now y además Microsoft se asegura de que es capaz de actualizar a Cortana de forma continuada por lo que mejorara más rápidamente. Cortana es un asistente de personalidad atrevida, de esta forma anima al usuario a que sienta que puede hablar con ella. Gracias a esto, Microsoft aumenta sus posibilidades de aprender lo que realmente quieren los usuarios, además Cortana utiliza estos datos mejorando continuamente, lo cual, será crucial es su futuro y en el de la imagen de la compañía.
Microsoft argumenta que su tecnología es lo suficientemente sólida como para que Cortana soporte un rango de tareas relativamente amplio sin comprometer su fiabilidad. Además, el modo que esta se conecta a la infraestructura de su motor de búsqueda Bing, plantea la posibilidad de grandes avances en la capacidad de entendimiento de la propia aplicación. Actualmente se están utilizando unos 300.000 ordenadores para procesar los datos web en bancos de memoria de Bing y encontrar las formas de extraer los conocimientos automáticamente para permitir a Cortana responder a nuevos tipos de consultas. Microsoft presentó la idea de que Cortana es un asistente mucho más complejo que los que existen actualmente, ya que no es solamente un programa que reaccione ante solicitudes mediante voz, sino que es capaz de recopilar información de

diversas fuentes como localización, redes Wifi, horas, recorridos, búsquedas. Por otro lado, también es capaz de tomar decisiones por el usuario e incluso tomar iniciativas, como pueden ser avisos de tiempo, cambios de estado según la localización o acciones según el número que llame. Con esto se confirma, que al asistente se le puede dotar de una mayor inteligencia ya que las variables de entrada van a ser mucho mayores.

2.2.4 Comparativa

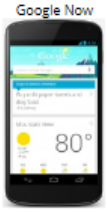
	Cortana	Siri	Google Now
			
Abrir aplicaciones	Si	Si	Si
Previsión del tiempo	Si	Si	Si
Calendario	Si	Si	Si
Configuración de alarmas	Si	Si	Si
Escribir consultas	Si	No	Si
Recordatorios	Si	No	No
Acceder a las funciones de las aplicaciones	Si	No	No
Llamadas	Si	Si	Si
Envío de mensajes o emails	Si	Si	Si
Reproducción de música	Si	Si	Si
Reconocimiento de música	Si	No	Si
Tecnología de búsqueda	Bing	Bing, Wolfram Alpha	Google
Sentido del humor	Si	Si	No

Figura 2: Comparativa Asistentes [6]

2.3 Expansión de la App al entorno *wearable*

2.3.1 ¿Por qué *wearables* y no otro entorno?

Según Dave Evans [10], internet ha ido evolucionando con el paso de los años pasando por un conjunto de fases hasta llegar al punto en el que nos encontramos hoy en día. Primero fue la fase de investigación, a esta la seguía la fase "brochureware", la cual está caracterizada por la “fiebre del oro” por los dominios Web. En la tercera fase la Web pasó de albergar datos estadísticos a proporcionar información transaccional gracias a la cual podían comprar y vender productos y servicios. Fue en esta etapa en la que surgieron empresas tan importantes como Amazon o eBay. Actualmente estamos en la cuarta fase, caracterizada por la predominancia de las Web “sociales” o de “las experiencias”, gracias a la cual empresas como Facebook, Twitter o Groupon son muy populares y rentables.

Internet of Things es la próxima evolución de internet. Como se ha visto, internet ha seguido un proceso continuo de mejora y desarrollo. Sin embargo, no ha cambiado demasiado hasta ahora. Con IoT se trata de dar un salto más, mejorando significativamente la manera de vivir, aprender, trabajar y entretenerse de las personas. También proporciona a internet la capacidad sensorial gracias a la gran cantidad de sensores que contienen estos nuevos dispositivos. Esto hará posible ser más productivos y menos reactivos.
Por otro lado, gracias a estos sensores tendremos la capacidad de expandirnos a lugares inimaginables hasta hace poco. Un claro ejemplo de esto son los pacientes que ingieren dispositivos que ayudan a diagnosticar y determinar las causas de ciertas enfermedades. Actualmente, se pueden integrar sensores extremadamente pequeños en plantas, animales o incluso accidentes geográficos y conectarlos a internet. Sin embargo, aún queda bastante para conseguir este nuevo mundo donde todo está conectado fundamentalmente debido a 3 elementos:

1. La implementación de IPv6 debido a la falta de direcciones ya que cada sensor requiere de una dirección IP única.
2. La energía con la que se alimentan los sensores ya que es necesario que estos sean autosuficientes.
3. Una mayor unificación en los estándares, especialmente en lo que a seguridad, privacidad y arquitectura se refiere.

Aunque todo este mundo de micro sensores aún queda lejano, sí que disponemos de un subtipo de estos dispositivos, los *wearables*:

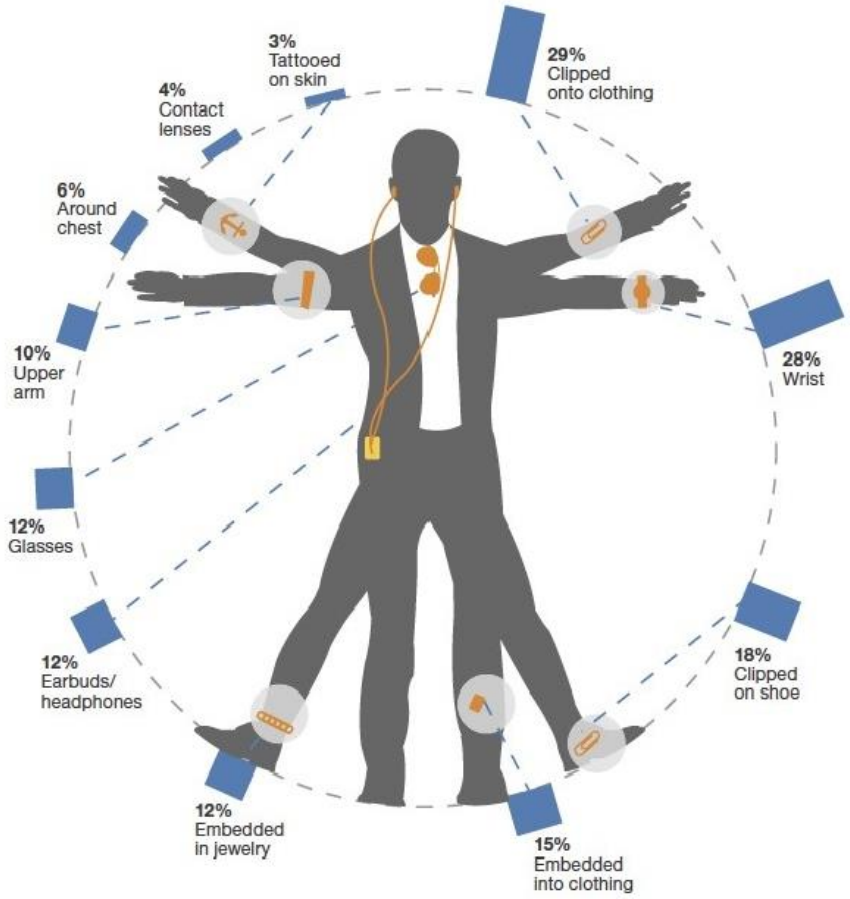


Figura 3: Tipos de *wearable* [7]

Como se puede observar en la imagen, hay un gran abanico de diferentes tipos de *wearables*, con el fin de adecuarse en la mejor medida a las necesidades del consumidor. En nuestro caso se pretende ampliar la funcionalidad de nuestra aplicación mediante un *wearable* que disponga de pantalla donde poder mostrar los datos, micrófono para poder hablar con el agente conversacional sin necesidad de sacar el móvil y capacidad de poder mostrar mapas. Si tenemos en cuenta estos factores el abanico de posibilidades se reduce a 2 dispositivos, reloj o gafas y de estos 2 dispositivos el que actualmente está más extendido es el reloj, por lo que la aplicación se realizara para este dispositivo.

Sin embargo, el hecho de que sea uno de los dispositivos más extendidos y que más tiempo llevan en el mercado no significa que aun este asentado. El mundo de los *smartwatches* aún se encuentra en fase beta. Todas las grandes compañías de la industria tecnológica están volcándose con estos productos y no hacen más que tantear y explorar diversas posibilidades. Según la página web hipertextual [8], como parte de esta experimentación Samsung presentó el Samsung Gear S, un nuevo *smartwatch* con Tizen cuya principal característica es la inclusión de conectividad 3G, algo que ya se había visto en algunas statups pero nunca en una de las grandes del sector y es que hasta hace poco todas las compañías estaban asumiendo que los *wearables* deben funcionar de forma conjunta con el smartphone. Sin embargo, desde la apuesta que hizo Samsung con su *smartwatch* cada vez son más las compañías que se hacen la pregunta ¿y si estamos siguiendo el camino equivocado?. La última gran empresa a sumarse al carro de los *wearables* independientes es LG con su Urbane 2 el cual, pese algunos contratiempos finalmente ha salido al mercado. Este dispositivo es el primero con conectividad 3G y sistema *AndroidWear*.

Teniendo en cuenta que la mayor parte de los *smartwatches* aún no tienen conectividad 3G se ha optado por elegir un *smartwatch* que comparta esta misma característica y que a su vez use el sistema *AndroidWear*. Por esto, después de observar los distintos *smartwatches* que había en el mercado optamos por comprar el siguiente:

2.3.2 *Smartwatch* LG GWatch

Especificaciones GWatch

Sistema Operativo	Android Wear 1.0 (compatible con Android 4.3+)
Correa	22mm - correa intercambiable
Pantalla	1.65" IPS LCD
Dimensiones	37.9 x 46.5 x 9.95 mm
Batería	400 mAh
Procesador	Procesador Qualcomm® Snapdragon™ 400 a 1.2GHz
Bluetooth	Bluetooth 4.0
Memoria	4GB de memoria interna y 512MB de RAM - La memoria disponible para el usuario es menor que la memoria interna total del dispositivo debido al espacio que ocupa el sistema operativo, el resto de funcionalidades y aplicaciones preinstaladas. La memoria disponible real para el usuario variará dependiendo de la versión de software instalada y de ofertas específicas vinculadas con terceros.
Puertos y conectores	Micro USB
Sensores	9 Axis (Accelerometer/Compass/Gyro)

Figura 4: Especificaciones del *smartwatch* elegido para el proyecto

3 Estructura de la aplicación a realizar

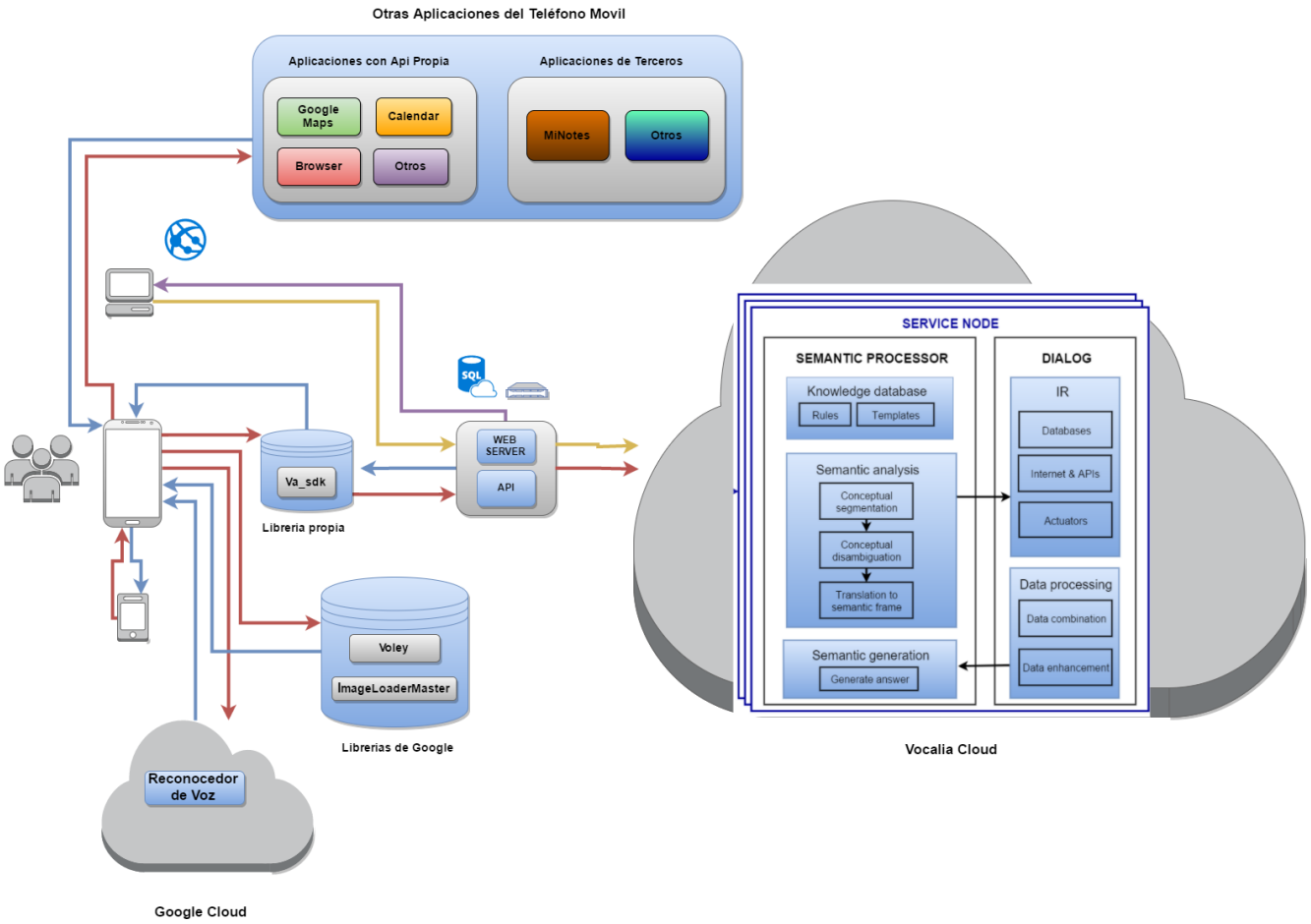


Figura 5 : Estructura de la aplicación a realizar

Como se puede observar en el gráfico de la figura 5, el objetivo es crear una aplicación con las siguientes capacidades:

1. Comunicarse con la nube de Vocalia Technologies, donde se encuentra la base de conocimiento del asistente virtual.
2. Comunicarse con la nube de Google para realizar el reconocimiento y síntesis de voz.
3. Comunicarse con aplicaciones con API de Google propia o de terceros.
4. Comunicarse con el dispositivo ***wearable*** y otorgar a este una mayor funcionalidad a la hora de mostrar la información.

Para poder otorgar a la aplicación este conjunto de funcionalidades fue necesario usar algunas librerías de Google e incluso crear una librería propia.

4 Plataforma en la nube

La infraestructura del asistente virtual esta implementada en la nube como un SaaS (Software as a Service), es decir, disponemos de un modelo de distribución de software en el que las aplicaciones están alojadas mediante un proveedor de servicio y disponibles para el usuario a través de una red (en nuestro caso como en la mayoría, es Internet). Este modelo de computación en la nube presenta una serie de ventajas:

- **Mayor agilidad:** Ya que posee la capacidad de mejorar los recursos tecnológicos que ofrecemos al usuario de forma más rápida.
- **Menor coste**: Un modelo de prestación pública en la nube convierte los gastos de capital en gastos de funcionamiento.
- **Mayor escalabilidad y elasticidad**.
- **Independencia entre el dispositivo y la ubicación** (siempre y cuando haya internet).
- **Mayor Rendimiento**: Los sistemas en la nube controlan y optimizan el uso de recursos de manera automática.
- **Seguridad:** Se puede mejorar debido a la centralización de los datos.
- **Mantenimiento más sencillo**.
- **Integración probada de servicios Web**: La tecnología ***cloud computing*** se puede integrar con mucha mayor facilidad y rapidez con el resto de aplicaciones empresariales (con o sin ***cloud computing*** ya sean desarrolladas de forma interna o externa).
- **Prestación de servicios a nivel mundial**.
- **Actualizaciones automáticas** que no afectan negativamente a los recursos de TI (Tecnología de la Información).

4.1 Arquitectura

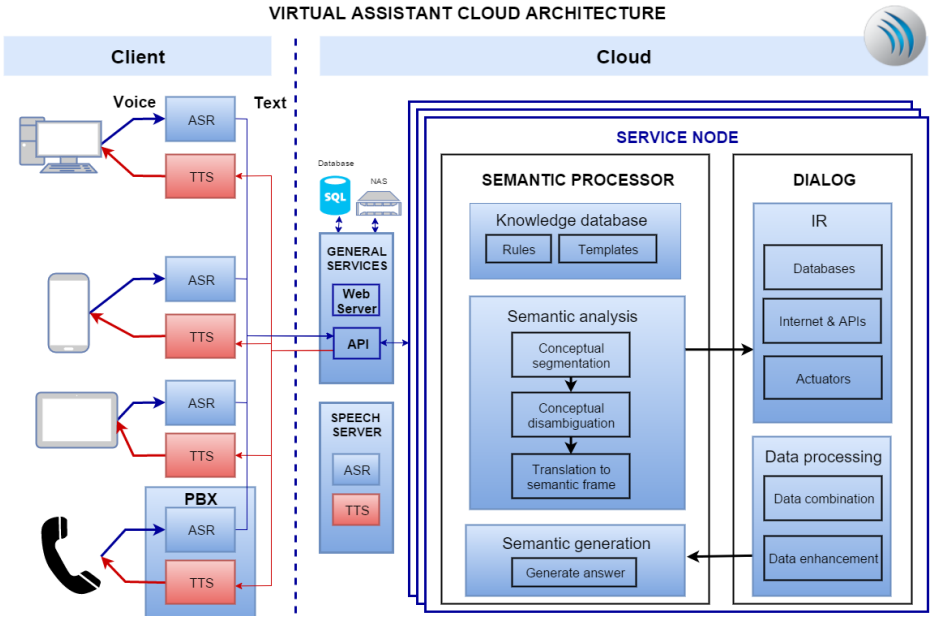


Figura 6 : Arquitectura en la nube del asistente virtual

Para poder realizar correctamente la aplicación Android hay que saber cómo comunicarse con el servicio SaaS. Para ello, a continuación, se expondrá el recorrido que realiza un paquete de datos desde la aplicación móvil hasta la nube.

Primero se realizará una pregunta al asistente. Para ello, se analizará el audio mediante el módulo ASR (Automatic Speech Recognition). Como se puede observar, el módulo ASR se encuentra en la parte del cliente, por lo tanto es a libre elección del programador. En este caso, como módulo ASR se usa el que provee Google (modificado para que no aparezca su interfaz gráfica).

Una vez que la pregunta ha sido realizada al asistente y se ha convertido a texto, se enviará al servidor web por medio de su API en formato JSON *(JavaScript Object Notation)*.
Por otra parte, en la nube se dispone de un conjunto de nodos de servicio, en los que hay un número n de motores AIML. La API decide a que nodo de servicio y a que motor AIML le envía la información. Además introduce una flag al inicio para indicar que se trata de unos nuevos datos a procesar por el módulo AIML elegido.
La API elige el nodo de servicio y el motor AIML mediante la clave introducida durante la conexión, ésta hace referencia a una ontología y a un idioma. En nuestro caso el idioma siempre será español y la clave viene indicada en el fichero de configuración.
Ahora que ya se sabe en qué nodo de servicio y a que motor AIML se accede dentro de ese nodo, se procederá a ver con más detalle los módulos que componen el nodo de servicio, entre ellos el ya visto motor AIML:

1. **Base de datos**: Almacena el conjunto de reglas y los *templates* semánticos. En ella se busca la información que ha pedido el usuario y ha sido procesada por el motor AIML.

2. **Motor AIML:** Se realizan 3 acciones diferentes:
    a. Análisis semántico.
    b. Procesamiento de datos.
    c. Generación semántica.

3. **Modulo IR (Information retrieval):** Módulos para otro tipo de búsqueda de información. Gracias a estos módulos se puede dar una mayor funcionalidad al asistente virtual, algunos de estos módulos son el WeatherServer, el cual provee de información relativa al tiempo atmosférico o el GoogleSearchServer para la búsqueda de información en internet.

Pero aún con todo esto, no es fácil ver cómo los diferentes módulos del nodo de servicio interactúan entre sí para finalmente mandar a la API una respuesta. En parte, esto es debido a que aún necesitamos saber cómo funciona AIML.

4.2 ¿Qué es AIML?

AIML (Artificial Intelligence Mark-up Languaje) es un lenguaje de programación de IA basado en XML. Su función es construir reglas de procesamiento de LN (lenguaje natural), para tratar las preguntas y generar las respuestas.

En este proyecto se utiliza AIML 2.0 para construir reglas que rigen el comportamiento del asistente virtual. Estas reglas se componen de 6 campos:

- **Activation count**: Contador de veces que se ha usado una norma.
- **Entrada:** Suele coincidir con lo que dice el usuario y será procesada para su posterior salida.
- **That:** Lo último que ha dicho el asistente virtual al usuario. El valor por defecto será * (cualquier valor desde una palabra a una frase entera).
- **Topic:** Valor asociado a la entidad, de esta forma el asistente virtual por medio de una serie de palabras clave reconocerá la entidad o *topic* y sabrá el tema sobre el que se está hablando.
- **Acción de salida:** Corresponde a la respuesta tras el procesamiento de la entrada.
- **Nombre del fichero:** Nombre del archivo donde se encuentra la regla en el momento de depurar.

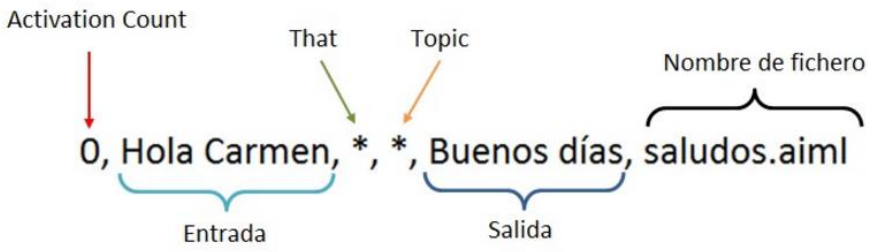


Figura 7 : Estructura básica en AIML

4.3 Interacción entre los Módulos

En el parser de AIML utilizado no se usa la técnica básica de análisis de frases de izquierda a derecha. Gracias al uso de herramientas primitivas se aumentó la complejidad a un sistema de parser con técnica híbrida de "***Island Driving***" o "***Island Growing***" junto con el análisis básico. Con esta técnica, según indica A.F. van Woudenberg [11], se detecta en primer lugar la entidad principal mediante ***Island Driving***. Posteriormente se analiza la frase por trozos con el análisis de izquierda a derecha, lo que va delante de la entidad, la entidad, y lo que viene después de la entidad.

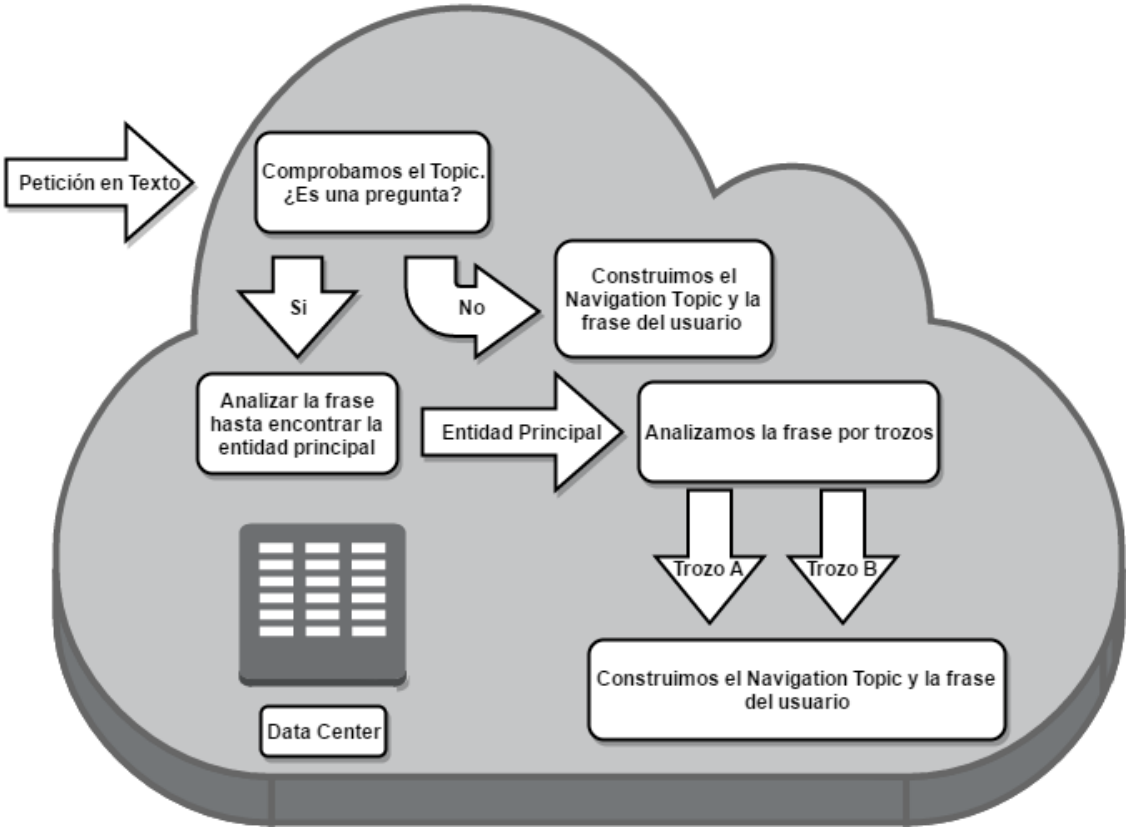


Figura 8: Interacción de los Módulos AIML

Como se puede ver en el esquema de la Figura 8, el proceso se divide en una serie de etapas:

1. El usuario le hace una petición al asistente virtual, ésta petición se convierte a texto y se pasa al módulo AIML.
2. Para analizar la frase, primero se hace una comprobación del *topic* para decidir si lo que está diciendo el usuario es una respuesta a una pregunta que se hizo anteriormente y si esta tiene información útil.
3. En caso de no ser contestación a una pregunta del asistente virtual, el primer paso para analizar la frase es encontrar la entidad principal.
4. Detectada la palabra clave, se analiza la frase por trozos.
5. A continuación, se analizan los distintos trozos de la petición del usuario por separado.
6. Finalmente, una vez analizada la petición del usuario, se construye el *NAVIGATION* utilizando el *topic*, que se le acaba de asignar, y la frase del usuario.

Ahora que ya se sabe cómo funciona el sistema con el que se comunica la aplicación para móvil, se procederá a explicar cómo se realiza la integración del asistente virtual con este dispositivo.

5 Desarrollo de la aplicación para móvil

5.1 Integración del asistente en el dispositivo móvil

El primer paso es el diseño de las funcionalidades de la aplicación en base a los requisitos especificados (conexión con las nubes de Vocalia y Google).

La aplicación para móvil permitirá realizar todas las operaciones que se puedan hacer desde la página web de la empresa, estas operaciones se realizarán mediante lenguaje natural. De esta forma, el usuario reduce el tiempo de interacción, mejorando su experiencia de uso y aumentando su nivel de satisfacción. Al no estar diseñada la aplicación para una empresa en concreto es necesario realizarla de forma modulable y escalable. Gracias a esto, la aplicación permite cambiar la ontología desde el menú de usuario. Por otro lado también será capaz de ajustarse a la imagen corporativa de cada empresa (colores, tipo de letra, etc.).

Esto conlleva una dificultad añadida, mostrar de forma óptima un conjunto de elementos que no comparten los mismos campos. Además, la interfaz ha de ser suficientemente atractiva e intuitiva para el uso del usuario. Por todo ello, para mostrar los diferentes elementos se usarán *CardViews*.
Sin embargo, el uso de *CardViews* también tiene inconvenientes. Al ser muy novedosos (surgen con la API versión 21, versión de Android 5.0 Lollipop, el 3 de noviembre de 2014), el porcentaje de dispositivos que será capaz de usar la aplicación a pleno rendimiento, es decir mostrando la información con *CardViews*, será menor, en torno al 35.4%, el resto verán la información como una lista normal de elementos.

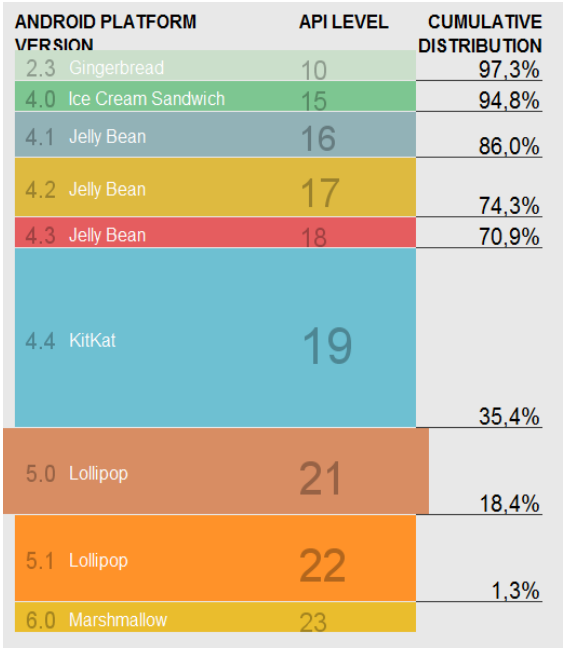


Figura 9: Porcentaje de uso de las versiones de Android

También se añadirán otras funciones como introducir mediante la API de Google la aplicación de Google Maps.

5.2 Diseño de la aplicación

Siguiendo las recomendaciones de Google, la aplicación se realizó de acuerdo con los cánones de Material Design con el fin de hacer la aplicación lo más intuitiva posible.

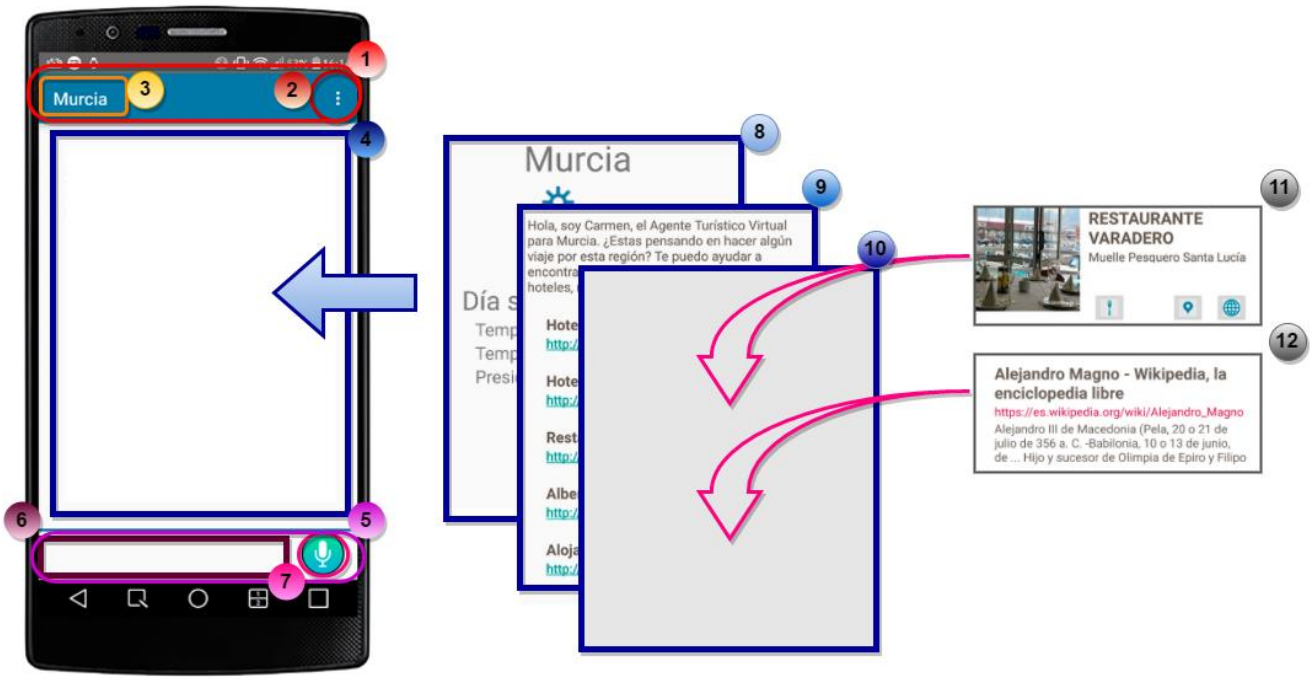


Figura 10: Diseño de la aplicación móvil

1. **ToolBar:** Diseñada como indican los cánones de Material Design. Contiene el título del asistente virtual en uso y un menú de configuración.
2. **Menú:** En él aparecen el conjunto de asistentes virtuales disponibles, cada uno con su propia interfaz visual ajustada a la imagen corporativa de la empresa asociada al asistente en cuestión.
3. **Título:** Muestra el asistente virtual en uso.
4. **Contenedor de Fragmentos[2]:** En esta aplicación se dio una mayor importancia a la información que devuelve el asistente. Es por ello por lo que este apartado ocupa la mayor parte de la pantalla. En él se cargaran los diferentes fragmentos según la información que vayamos a mostrar.
5. **Parte inferior de la pantalla del dispositivo.**
6. **Botón encargado de grabar las peticiones del usuario.**
7. **Espacio destinado a mostrar la respuesta del asistente.** Debido al reducido tamaño que se dispone se optó por mostrar el texto en cuestión en formato marquesina (texto en movimiento).
8. **Fragmento destinado a mostrar el tiempo.**
9. **Fragmento destinado a mostrar FAQ.**
10. **Fragmento contenedor de *CardViews*.** Se diseñó con un fondo más oscuro para destacar las diferentes tarjetas.
11. **CardView destinada a mostrar información con foto.** Permite acceder a mapas o lanzar un *browser* predeterminado.
12. **CardView destinada a mostrar información sin foto.** En este caso se realizaron las tarjetas clicables. Al clicar la tarjeta se lanzará el *browser* predeterminado del dispositivo, o dará a elegir el *browser* que se quiere utilizar.

[2] Llamamos fragmento a las clases que extienden de ***Fragment***. Un fragmento en Android representa un comportamiento o una porción de la interfaz de usuario en una actividad.

5.3 Información de entrada

En esta aplicación se dispone de 2 vías diferentes de obtener información:

- **Fichero de configuración:** Se encuentra en la capeta ***raw*** [3]dentro de los recursos de la propia aplicación. La finalidad de este fichero es dar forma a la propia aplicación y decidir las funcionalidades e imagen corporativa que se quiere para cada asistente virtual.
- **La nube**: En la nube disponemos de la base de conocimiento del asistente virtual la cual nos enviará la información pedida por el usuario mediante una estructura JSON. Esta estructura se repetirá independientemente del asistente virtual al que se pregunte.

5.3.1 Fichero de Configuración

A continuación se muestra un ejemplo sencillo, con una única ontología en el que se puede ver la estructura (también de tipo JSON) del fichero:

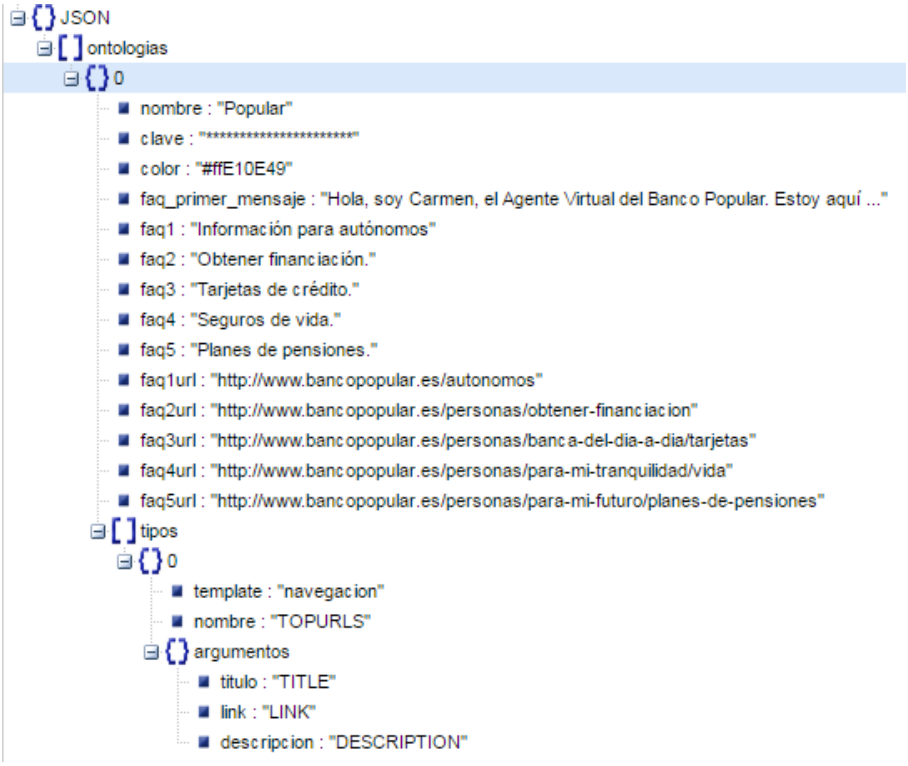


Figura 11: Ejemplo Fichero de Configuración

[3] En Android es necesario crear una carpeta raw para trabajar con ficheros de audio, video o archivos de texto. Esta carpeta tiene que estar ubicada en ./res.

Como podemos observar, disponemos de un objeto JSON el cual contiene un array de ontologías (para simplificarlo hemos puesto solo una de ellas). Esta ontología dispone de una serie de campos: nombre, clave, color, las preguntas más frecuentas (las cuales usaremos para la interfaz visual de inicio) y los tipos. En este caso solo disponemos de un tipo llamado TOPURLS al que asociamos un determinado formato de *CardView* mediante el *template* y los argumentos con los que rellenaremos estas *CardViews*. En este caso cada *CardView* dispondrá de un título, un link a una página web determinada y una descripción del contenido que hay en esa página web.
En el caso real disponemos de 3 ontologías distintas: Murcia, Banco Popular y ESA, las cuales están formadas por los mismos campos pero cada una de ellas tiene diferentes tipos de datos, con unos argumentos y una *CardView* asociada.

5.3.2 Información de la nube

Para mostrar esto de la mejor manera posible, analizaremos un ejemplo real. La respuesta que nos da el asistente virtual a la pregunta "muéstrame los monumentos que hay en Murcia":

Nota: El código tenía 4 datos pero con mostrar solo 1 de ellos vale el resto fueron borrados por eso sale NUM_RESULTS = 4 aunque solo aparezca uno de ellos.

Figura 12: Ejemplo JSON proveniente de la nube

Para analizar la respuesta se explicarán los diferentes campos que la componen y los diferentes valores que pueden tomar esos campos:

- **DATA**
  - **FOUND_RESULTS *[“TRUE”/ “FALSE”]*:** Informa si la pregunta ha generado datos como respuesta, si es *TRUE* ha encontrado uno o más resultados, en caso contrario será *FALSE*.
  - **TYPE** [“ALOJAMIENTO”/ “RESTAURANTE”/ “PLAYA”/ “MONUMENTO”/ “ELEMENTO”/ “SEARCH”]: Muestra el tipo de dato devuelto.
  - **NUM_RESULTS:** Indica el número de datos obtenidos.
  - **RAW_DATA:** Lista de elementos encontrados por el asistente virtual. Hay que tener en cuenta que solo será distinto de *null* para oraciones resolutivas (ej. muéstrame las playas). Si por el contrario son preguntas coloquiales (ej. Hola, que tal estas, ¿para qué sirves?) valdrá *null*. Por otro lado, cada elemento dentro del campo datos tendrá sus propios argumentos dependiendo del tipo de dato que nos devuelva.
- **TTS:** Contiene la respuesta del asistente virtual en *formato UTF-8.*
- **INPUT**
  - **TEXT:** Contiene la pregunta realizada al asistente virtual en *formato UTF-8.*
  - **UNDERSTOOD *[“TRUE”/ “FALSE”]*:** Si ha entendido la pregunta valdrá *TRUE*, en caso contrario valdrá *FALSE*.
- **NAVIGATION (*NAV*)**:
  - **STATUS *[“OK”/ “ERROR”]*:** Si vale *OK* una o más urls son accesibles, por el contrario, si vale *ERROR* la navegación, ya sea directa o asistida ha fallado.
  - **TYPE [“*DIRECT*”/ “*ASSISTED*”]:** Si es *DIRECT* devuelve una única url, en caso de que sea *ASSISTED* devuelve los primeros 10 resultados obtenidos en el buscador.
  - **URL**: Devuelve la url de la página principal ya sea la navegación directa o asistida.
  - **TOP_URLS**: Este campo no es nulo sólo en caso de navegación asistida. El objeto devuelto sigue el siguiente formato:


```
"TOP_URLS": [
 {
      "TITLE":"Archena: Web oficial turismo Región de Murcia",
      "LINK":"http://www.murciaturistica.es/es/archena/",
      "DESCRIPTION":"Archena es un municipio de la Región de Murcia, situado en el Valle de Ricote y Cieza, conocido por sus aguas termales."
 },
```

{

"TITLE":"La Manga: Web oficial turismo Región de Murcia",

"LINK":"http://www.murciaturistica.es/es/la_manga/",

"DESCRIPTION":"La Manga es uno de los iconos turísticos de la Región de Murcia. 24 kilómetros de tierra rodeados por dos mares: Mediterráneo y Mar Menor, con playas de..."

}, {...}]

- **FAQ ["*TRUE*"/ "*FALSE*"]:** Devuelve en el caso de que la pregunta no sea resolutiva un fichero con las FAQ. Sin embargo, esta funcionalidad no está operativa y siempre vale *FALSE* por lo que pongo las FAQ en el fichero de configuración a mano como ya se vio anteriormente.

5.4 Diseño de la Aplicación

5.4.1 Estructura

La aplicación se divide en 2 partes:

5.4.1.1 Librerías

5.4.1.1.1 Externas

ImageLoaderMaster

Esta librería permite mostrar correctamente las imágenes. Para ello, una vez que se ha obtenido la dirección de la imagen con ayuda de esta librería, se descarga la imagen, se decodifica en formato bitmap y por último, se muestra en los ***ImageViews*** del ***adapter*** de Murcia.

Volley [9]

Según James Revelo [9], Volley procesa el envío de peticiones Http. Esta librería ha sido creada por Google para optimizar el envío de peticiones Http desde las aplicaciones Android a los servidores externos. Por otra parte, esta librería está enfocada completamente en las peticiones. De esta forma, se crea código repetitivo por cada petición o incluso a la hora de parsear los datos.

Características Principales

- Procesamiento concurrente de peticiones.
- Priorización de peticiones.
- Cancelación de peticiones, evitando la presentación de resultados no deseados en el hilo principal.
- Gestión automática de todos los trabajos en segundo plano.
- Implementación de cache en disco y memoria.
- Capacidad de personalizar las peticiones.
- Provee información detallada del estado de flujo de trabajo de las peticiones en la consola de depuración.

- De todas estas ventajas únicamente interesan 3 de ellas:
  1. Gestión automática de todos los trabajos en segundo plano.
  2. Capacidad de personalizar peticiones, "la cual se usará para meter una ***sessionCookie*** como dato en las cabeceras de las peticiones". Con esta ***sessionCookie*** el asistente virtual en la nube podrá saber cuándo una petición forma parte de una conversación previa y cuando no.
  3. Capacidad de proveer información detallada del estado de flujo de trabajo de las peticiones en la consola de depuración. Esto es de gran utilidad a la hora de comprobar errores.

- Como contra, no es buena para la descarga de datos pesados. Sin embargo, en este proyecto nunca se tendrán que descargar grandes cantidades de datos, por lo que no hay problema.

5.4.1.1.2 Propias

Va_sdk

Esta librería tiene la finalidad de separar toda la funcionalidad relacionada con el asistente virtual del resto de la aplicación, de esta forma en el caso de que otra persona quisiera usar las funcionalidades del asistente solo tendría que descargársela. Está formada por 5 clases y 3 interfaces cuya finalidad y métodos serán explicados a continuación:

ApiConnector

Esta clase es la encargada de crear las preguntas (en formato texto), enviar las preguntas a una cola de salida y que desde ahí, automáticamente, se envíen a la API del asistente virtual.

ApiRequest

Esta clase recoge las respuestas provenientes de la API del asistente virtual y comprueba la cabecera de la respuesta en busca de la ***sessionCookie.*** En el caso de que se encuentre, la guarda.
Cabe destacar que, cuando se recogen las respuestas con la librería Volley, ésta llama al método ***HttpResponse*** de la clase ***HttpClientStack*** el cual, internamente, añade la cabecera cogiendo la que tenga la última respuesta analizada, llamando a ***getHeaders***.
Para conseguir añadir una cabecera a una respuesta, se sobrescribe el método ***getHeaders*** anteriormente comentado. De esta forma, en el caso de que no tenga cabecera la respuesta, se crea un campo *header* dentro de la misma. Por último, al final del método, se añade la *cookie* de la sesión.

VirtualAgent

Es la clase principal de ***Va_sdk***. En ella se encuentran todos los métodos que se pueden llegar a necesitar en el caso de que se quiera usar esta librería. Dentro de este proyecto no se han llegado a usar todos los métodos creados en la misma. Sin embargo, estos podrían ser de utilidad para otros usuarios que la usen. Los métodos implementados son los siguientes:

- Inicialización del SDK.
- Comprobación de si se tiene el reconocedor de voz de Google instalado en el terminal.
- Enviar petición a la API del asistente virtual.
- Enviar audio a Google para que lo reconozca y envíe directamente el texto reconocido por Google a partir del audio enviado a la API del asistente virtual.
- Escuchar un texto por el altavoz del teléfono.
- Escuchar la respuesta del asistente virtual por el altavoz del teléfono.
- Obtener si está activo el ***AutoPlay*** de las respuestas por audio.
- Activar/Desactivar el ***AutoPlay*** de las respuestas por audio.
- Obtener pregunta realizada al asistente virtual.
- Obtener la respuesta enviada por la API del asistente virtual.
- Obtener si el asistente virtual ha entendido la pregunta.
- Obtener la versión de la API del asistente virtual.
- Obtener el JSON completo devuelto por el asistente virtual.
- Obtener si la pregunta ha generado una o varias urls de navegación válidas.
- Obtener la url de la navegación.
- Obtener si ha generado un listado de posibles urls de navegación.
- Obtener si ha obtenido un conjunto de datos como respuesta a la pregunta realizada.
- Obtener el tipo de datos que ha generado la pregunta realizada.
- Obtener el número de datos que ha generado la pregunta realizada.

- Obtener el array de objetos JSON que contienen los datos que ha generado la respuesta realizada.

VoiceProcesor

Esta clase contiene todo lo necesario para poder reproducir cualquier texto mediante audio en el dispositivo móvil. A ella acceden los métodos de la clase ***VirtualAgent*** cuando quieren convertir un texto en voz y reproducirlo por los altavoces del terminal. Además, en el método de inicialización del SDK de la clase ***VirtualAgent*** se introduce como atributo el idioma con el que tiene que reproducir el audio. Toda la conversión de texto a audio se realiza mediante Google. El acento con el que suena en el teléfono móvil lo puede elegir el usuario cambiando la configuración de su teléfono móvil en el apartado de ajustes.

VolleySingleton

Esta clase es la encargada de restringir la cantidad de preguntas que se envían a la API del asistente virtual a una cada vez.

5.4.1.2 Aplicación

Para explicar el funcionamiento de la aplicación se expondrá de forma individual el funcionamiento de cada clase y la interconexión que hay tanto entre las diferentes clases de la propia aplicación como entre las clases de la propia aplicación con las clases de las librerías.

MainActivity

Es la clase principal y contiene los siguientes métodos:

- ***OnCreate:*** Primer método al que se accede al ser creada la clase.
  - Se asigna a esta clase una vista "layout" inicializando los elementos que la componen.
  - Se piden los permisos necesarios al usuario para el correcto funcionamiento de la aplicación.
  - Se inicializa el asistente virtual indicando que la comunicación se realizará en español.
  - Al pulsar el botón **grabarAudio** se inicializa el ***VoiceListener*** (encargado de grabar audio y convertirlo en texto).
  - Se crea una nueva instancia de la clase abstracta ***ApiConnectionListener*** asociada a la librería ***Volley***, en la que se sobrescribe el método ***onSuccess***. Este método, se llama cuando la comunicación tiene éxito. Es en este método donde se recoge el JSON que nos envía el asistente virtual. Este

JSON tiene que ser leído para saber qué tipo de datos contiene y crear el fragmento adecuado en función del tipo de información a representar.

- ***OnPrepareOptionsMenu*:** Crea un menú en la parte superior derecha que contiene todas las ontologías indicadas en el fichero de configuración.
- **OnOptionsItemSelected:** Permite al usuario seleccionar el asistente virtual al que se realizarán las preguntas. Con ello se modifica la estética del layout en función de la ontología elegida (según los parámetros introducidos en el fichero de configuración).
- **OnSelector**: Sustituye el fragmento creado por el que se encuentra en el contenedor de fragmentos.

5.4.1.2.2 Clases destinadas al análisis del fichero de configuración

OntologySelector

Esta clase contiene los siguientes métodos:

- ***SetOntology***: Modifica la ontología del programa por otra que le pasen como argumento. Se le llama cuando alguien elige una ontología dentro del menú de la ***MainActivity***.
- ***GetMapOntology:*** Devuelve los posibles tipos de ontologías donde poder elegir. Se usa para crear el menú de la ***MainActivity***.
- **ReadDocuments**: Método llamado desde el constructor. Su finalidad es leer el fichero de configuración, transformarlo en un objeto JSON y pasárselo al método ***ParserConfig*** de la clase ***Parser***.

También contiene 2 métodos no relacionados con el fichero de configuración: ***GenerateInfoView*** y ***GenerateView*** los cuales llaman a los métodos ***generateInfoView*** y ***generateView*** de la clase ***Ontologia*** y devuelven el fragmento creado.

Parser

Esta clase contiene un solo método:

- **ParserConfig:** Extrae del ***JsonObject*** el ***JsonArray*** que contiene todas las ontologías. Extrae de este ***JsonArray*** las ontologías y las introduce en un objeto ***map***. Cada una de estas ontologías será analizada en la clase ***Ontologia*** mediante el método ***ParserObject***.

Ontologia

Esta clase contiene un método relacionado con el fichero de configuración:

- **ParserObject**: Este método saca todos los campos de un ***JsonObject***, en este caso todos los campos de la ontología. Entre estos campos se encuentra el campo tipos, que no es un campo final sino un ***JsonArray***. Se recorre el ***JsonArray*** de tipos cogiendo cada tipo y se pasa al método ***parserTipo*** de la clase ***TiposDatoOntologia***.

El resto de métodos están enfocados a la modificación del fragmento superior:

- **GenerateInfoView**: Crea un nuevo objeto InfoView.
- **GenerateView:** El tipo de dato puede ser:
    - **Tiempo**: Llama a la clase ***Weather*** y le pasa el ***JsonArray*** metiéndolo en un ***Bundle.*** Este ***Bundle*** se pasará como argumento del método ***setArguments*** de esa misma clase.

    - **Otros:** Crea un ***ListElemets*** y le pasa la plantilla "*template*", el tipo de argumentos y el ***JsonArray*** de los datos que se obtienen de la respuesta del asistente virtual mediante un ***Bundle.*** Este ***Bundle*** se pasará como argumento del método ***setArguments*** de esa misma clase.

TiposDatoOntologia

Esta clase contiene un solo método:

- **ParserTipo**: Extrae el *template*, el nombre del tipo y los argumentos del tipo. Con estos argumentos se cogen las claves o nombres que contienen el dato y se comprueba cuales son de icono. Con todos los iconos se crea un ***Hashmap***, el cual será pasado al método ***parserIcon*** de la clase ***Icono***. Con los argumentos que sean de tipo texto se hará un ***Hashmap*** similar.

Icono

Esta clase contiene un solo método:

- **ParserIcon**: Obtenemos el tipo de icono que se corresponde con la imagen que habrá que mostrar y a su vez con el nombre del icono.

5.4.1.2.3 Clases destinadas a mostrar la información que llega del fichero de configuración

InfoView

Esta clase es la encargada de mostrar las FAQ asociadas al asistente virtual con el que se comunica la aplicación. Es llamada en el método de la ***MainActivity onOptionsItemSelected*** con el fin de que, al cambiar de asistente virtual, el usuario pueda ver, en un primer momento, las preguntas más frecuentes que se le pueden realizar.

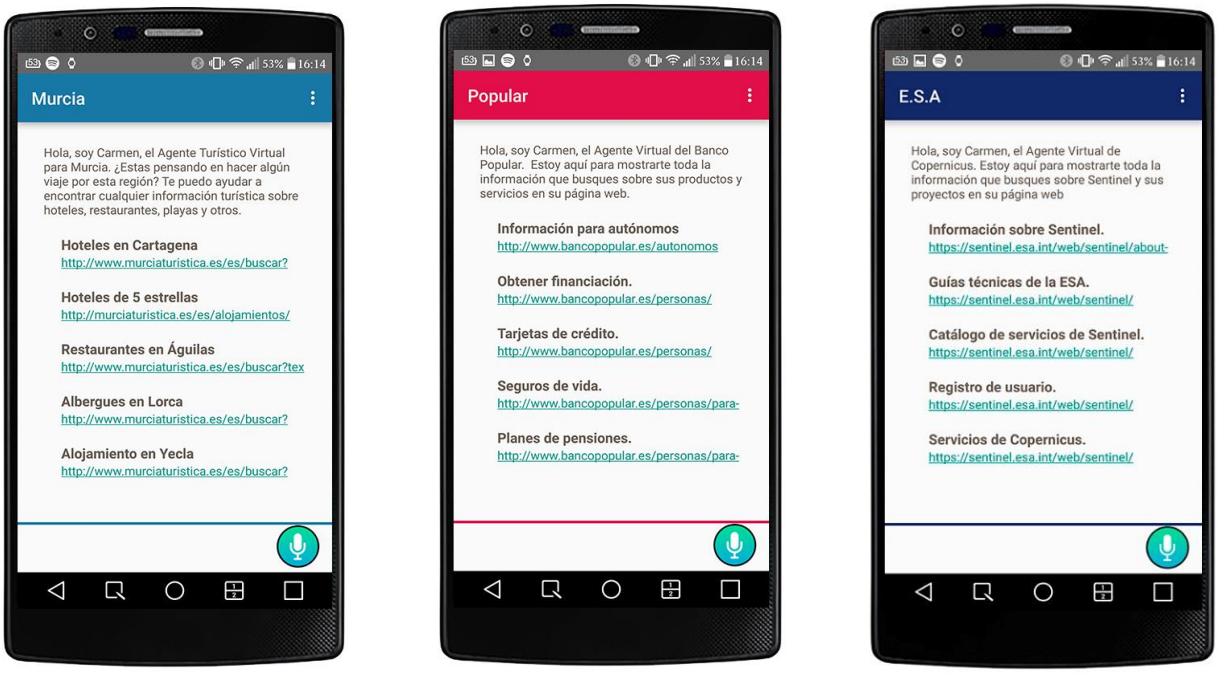


Figura 13: Todas las vistas de InfoView

5.4.1.2.4 Clases destinadas a mostrar la información que nos llega de la API del Asistente Virtual

Figura 14: Vista Weather

Weather

Esta clase es la encargada de mostrar la información asociada al tiempo de una forma atractiva para el usuario. Es el único fragmento junto con el de FAQ que no muestra información mediante *CardViews*. De esta forma se muestra en el fragmento los siguientes datos: ciudad, icono de tiempo, tiempo, temperatura media, temperatura máxima y mínima, presión atmosférica y humedad.

ListElements

Esta clase recoge el *template*, los argumentos del tipo de dato y un JsonArray formado por la información que proviene del asistente virtual. Llama al método ***ChooseAdapter*** el cual, según el campo *template* decidirá el tipo de *adapter* a usar llamando a las clases ***DownloadListTaskLayoutConFotoEIconos*** o ***DownloadListTaskTopNavigation*** según el tipo de *adapter* que se pretenda usar.

Posteriormente pasa al adapter seleccionado los argumentos que le llegaron a él ya transformados a *String*. En nuestro caso solo disponemos de 2 *adapters*:

- Con foto
- De navegación

DownloadListTaskLayoutConFotoEIconos/ DownloadListTaskTopNavigation

Estas clases son prácticamente iguales, la única parte que cambia es el *adapter* al que invocan por lo que se explicarán de forma conjunta.
Ambas clases extienden de ***AsyncTask <Void, String, Adapter>,*** esto hace que nada más ser ejecutadas desde la clase ***ListElements*** creen un hilo secundario al que le pasaran la vista, los datos de la respuesta del asistente virtual, los argumentos del tipo y el contexto. Estos argumentos se usaran para crear el adapter desde el hilo secundario. Cuando el adapter está creado se lanza el método ***PostExecute*** donde modificará la vista que le llegó por parámetro añadiéndole el *adapter* creado.

AdapterElementosNavigation

Esta clase extiende de ***RecyclerView.Adapter<AdapterElementosNavigation.ListViewHolder>.*** ***ListViewHolder*** la cual, es una clase anidada que contiene la clase ***AdapterElementosNavigation*** cuya view está asociada a una ***CardView*** sin foto. Tiene como elementos para rellenar dentro de esta ***CardView*** el título, el link a la página de internet y la descripcion del enlace.

- ***Constructor*:** Recibe como parámetros, la clase donde se genera el objeto *adapter*. En este caso, ***DownloadListTaskTopNavigation,*** los datos obtenidos de la respuesta del asistente virtual, los argumentos tipo y por último, el contexto. Todos estos datos serán usados en el método ***OnBindViewHolder***.
- ***OnBindViewHolder:*** Encargado de configurar el contenido de las *CardViews*. Se irán cogiendo los diferentes ***JsonObject*** que contiene el ***JsonArray*** de la respuesta del asistente virtual. De esta forma, cada *CardView* está asociada a un ***JsonObject***, por lo que solo hay que sacar los campos de ese objeto y colocarlos en los campos de la *CardView* donde correspondan. En nuestro caso estos campos a rellenar serían solo título, link y descripción.
- ***GetItemCount*:** Permite mostrar tantas *CardViews* como cantidad de ***JsonObject*** tiene la respuesta del asistente virtual.

También se ha implementado una opción para que la propia *CardView* tenga un evento de cliqueo (como si fuera un botón). De esta forma cuando se pulse nos redirigirá a la página web indicada en el link.

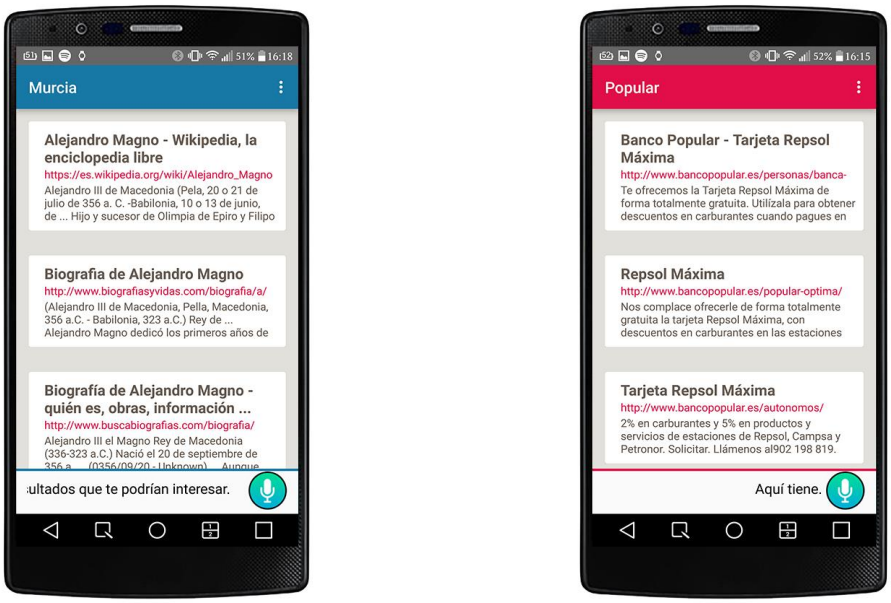


Figura 15: Vista de los datos de navegación

AdapterElementosMurcia

Esta clase extiende de ***RecyclerView.Adapter<AdapterElementosMurcia.ListViewHolder>***.La clase anidada ***ListViewHolder*** tiene su view asociada a una *CardView* con foto y tiene como elementos para rellenar dentro de esta *CardView*:

- Imagen
- Nombre
- Dirección
- Teléfono
- Otros
- Icono
- BotonUrl
- BotonMapa

A la hora de coger los datos es necesario poder diferenciar aquellos que son de tipo icono. Cuando llega un dato con tipo icono se crea un ***Hashmap*** para poder guardar los distintos iconos con sus atributos. Todos los iconos tienen como atributo "tipo" el nombre de la imagen que van a usar. Sin embargo, no todos tienen relleno el atributo ***IconParameter***. El atributo ***IconParameter*** se usará para indicar el campo que tiene que coger del JSON (proveniente del asistente virtual) para mostrar una información. En el caso de ***iconoUrl*** este campo contendría la dirección web, mientras que en el caso de hoteles, contendría el número de estrellas del hotel.

En iconos de categoría la imagen a mostrar será aquella que tenga como nombre la unión de los dos campos que contiene el icono.

Para el caso de que se trate de un dato que no sea de tipo icono también se crea un HashMap, esta vez exclusivo para datos que no sean iconos. De esta forma, se podrá acceder a ellos de forma simple a posteriori (usando las *keys* del HashMap[4]).

Otro apartado importante en este *adapter* es la forma en la que se rellena el campo **ImageView** del *CardView*. Ya comentamos anteriormente que se hace con ayuda de la librería **ImageLoaderMaster**. Sin embargo, ahora se verá en detalle:

1. Se comprueba que el campo foto contenido en el **HashMap argumentosTexto** no sea nulo.
2. Se hace visible la imagen en el *layout*.
3. Se obtiene una instancia Singleton y se decodifica la imagen en un formato Bitmap.
4. Si es la primera imagen se inicializan los parámetros de la librería **ImageLoader**.
5. Se muestra la imagen en el *CardView*.

A la hora de mostrar los datos en el *CardView* se hará de forma similar al anterior *adapter*, con la diferencia que en este se dispone además de 2 botones:

1. Redirige a página web
2. Abre el mapa.

Estos botones al usar métodos que llaman a APIs externas serán explicados detenidamente en un apartado independiente.

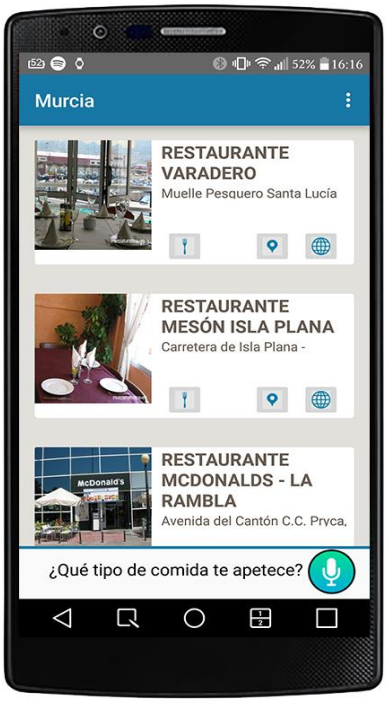


Figura 16: Vista de los elementos de Murcia

[4] Antes de poder introducirlos en el HashMap es necesario modificar el texto que contiene ya que no es capaz de interpretar acentos, ñ, etc. Por lo que se realiza un cambio de formato al texto.

5.5 Conectividad de la Aplicación con otras Aplicaciones

En este TFG se pretende mostrar la capacidad de un asistente virtual a la hora de controlar otras aplicaciones. Para ello, se ha llevado a cabo un proceso de búsqueda y análisis de las diferentes metodologías existentes.
El primer elemento a tener en cuenta es como se quiere acceder a estas aplicaciones. De esta forma hay 4 posibilidades:

1. **Lanzar una aplicación externa desde la aplicación del asistente.**

2. **Lanzar métodos de aplicaciones externas desde la aplicación del asistente**: Desde cualquier aplicación es posible lanzar métodos contenidos en aplicaciones de terceros, siempre que estas posean una API, sea o no la de Google. La única información necesaria es conocer con que métodos es posible interaccionar, los parámetros de entrada y salida de los mismos. Este caso tiene como inconveniente que el usuario no sabe si la operación requerida por un comando de voz se realizó correctamente.

3. **Lanzar métodos de aplicaciones externas desde la aplicación del asistente, junto con una interfaz propia para la correcta visualización de los datos**: Debido al diseño de la aplicación realizada, lo óptimo sería crear un nuevo fragmento que desempeñe la labor de visualización de los datos y sustituirlo por el que se muestra en el contenedor de fragmentos. Sin embargo, se comprobó que la aplicación perdía accesibilidad. Esto es debido a que el botón "atrás" en el móvil tiene como función interna pasar a la actividad anterior y esta no varía. Lo que varía, es el contenido del contenedor de fragmentos.

Por todo esto, se decidió en nuestro caso, que a la hora de interaccionar con otras aplicaciones (como bien puede ser el navegador de internet o el Google Maps) estas se lanzaran en vez de integrarlas en nuestra propia aplicación. Para ello, como ya vimos, creamos 2 botones:

1. **Botón de Navegación:** Lanza un *browser* con la url del elemento en cuestión.
2. **Botón de Mapas:** Lanza el Google Maps con el camino ya trazado desde donde nos ubicamos hasta el lugar donde se sitúa un elemento. Para poder llevar a cabo esta acción se tiene que crear un ***LocationManager*** que compruebe que el GPS del terminal está habilitado y luego, un ***LocationListener*** que vaya actualizando la posición del sujeto respecto del lugar al que se quiere ir.

El siguiente paso es crear un asistente virtual destinado a interaccionar con otras aplicaciones de forma que estas sean un fin, no un medio. Un claro ejemplo de esto sería introducir en la aplicación realizada la capacidad de interaccionar con la aplicación de Calendario. En este ejemplo, el asistente virtual será el encargado de preguntar hasta que consiga rellenar todos aquellos campos que el método ***crearEvento*** de la aplicación

Calendario tenga definido como obligatorios[5]. Posteriormente preguntará si se desea rellenar algún campo no obligatorio pidiendo la información necesaria para rellenarlo.

Una vez que el procedimiento destinado a recopilar información sobre la acción que se desea realizar finaliza, envía en formato JSON todos estos datos, los cuales serán necesarios a la hora de interaccionar con la aplicación de Calendario.

Como ya hemos visto anteriormente el proceso de interacción con otras apps (ya sean con API de Google o no) es completamente factible, por lo que no debería ser un proceso demasiado complejo.

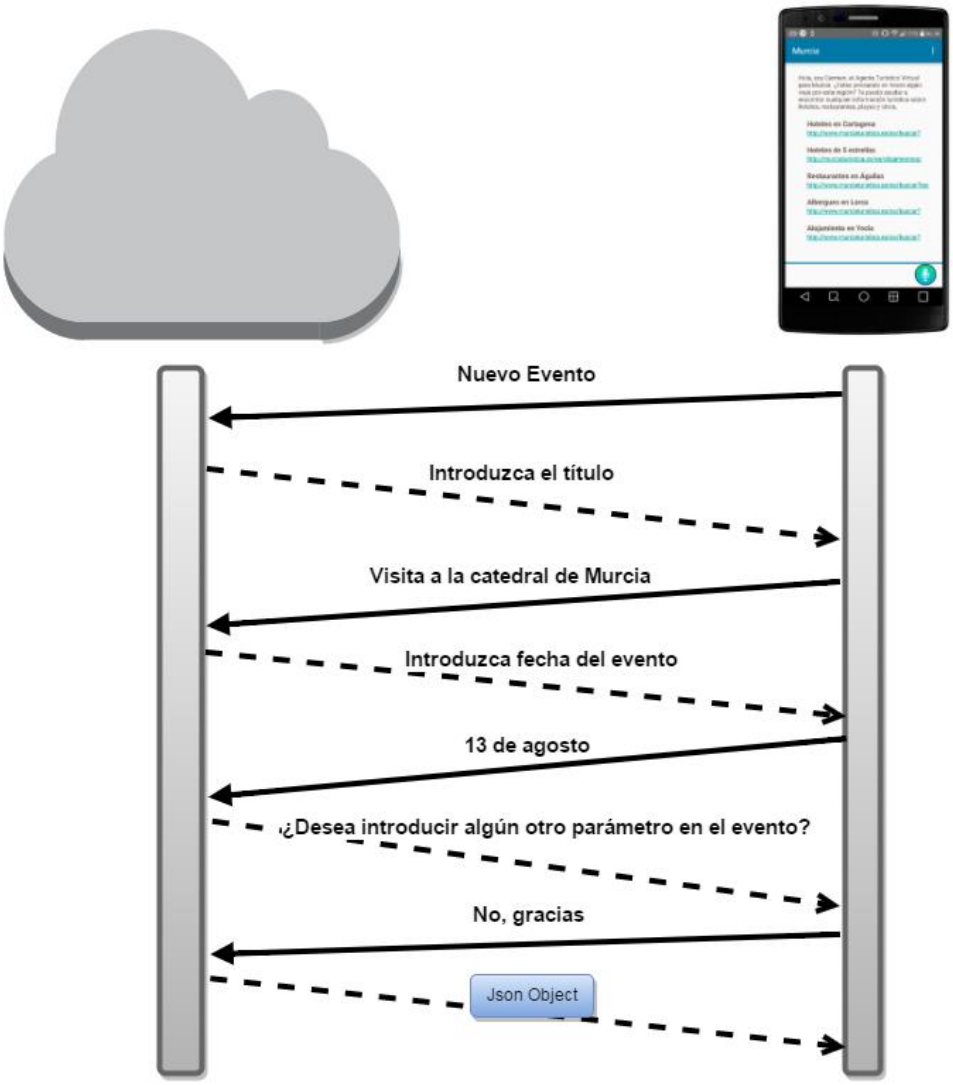


Figura 17: Diagrama de comunicación entre la App y el asistente virtual para añadir un evento al calendario

[5] La mente del asistente virtual debe contener todos los posibles campos a rellenar del método en cuestión con el que se quiere interaccionar. Una vez introducidos se indicaran que atributos son obligatorios (mediante el elemento *).

6 Desarrollo de la aplicación para *wearable*

Actualmente las aplicaciones para *wearables* que están en el mercado pueden ser de 4 tipos:

1. **Aplicaciones estéticas**: Estas aplicaciones tienen la finalidad de modificar la apariencia del dispositivo.
2. **Aplicaciones deportivas:** Estas aplicaciones usan los sensores que tiene el dispositivo para contar los pasos o algunas incluso marcar las calorías consumidas.
3. **Aplicaciones de productividad:** Ej. Aplicaciones con función de agenda, tiempo, alarma, etc.
4. **Aplicaciones basadas en notificaciones:** Ej. WhatsApp, esta aplicación cuando recibe un mensaje manda una notificación al *smartwatch*, mostrándola por pantalla.

La aplicación realizada propone algo distinto a lo que actualmente está en el mercado. A diferencia de otras aplicaciones, esta pretende ser una herramienta de interacción en sí misma. Por ello, se decidió no utilizar notificaciones, ya que estas restan usabilidad al dispositivo *wearable*, pudiendo resultar tedioso para el usuario.

6.1 Diseño de la aplicación para *smartwatch*

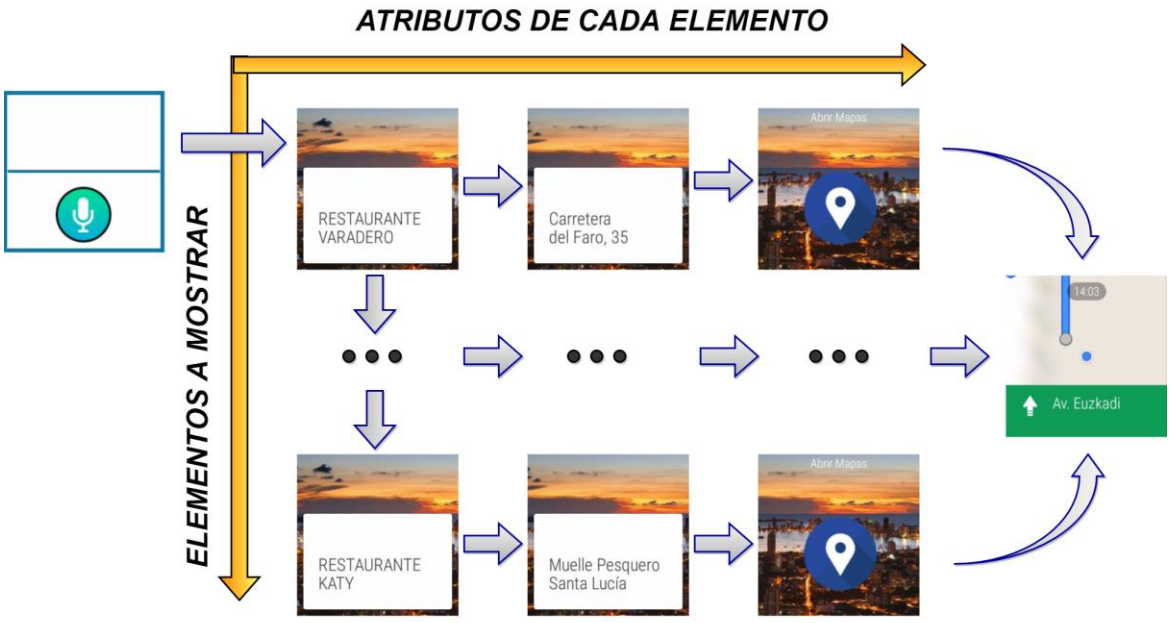


Figura 18: Diseño de la aplicación en el *smartwatch*

El diseño que aparece en la figura 18 es un diseño pensado para dar una funcionalidad completa a la aplicación. Con él se es capaz de realizar todo el proceso de comunicación sin salir del dispositivo:

1. El usuario pide mediante lenguaje natural unos datos desde la interfaz principal. Esta interfaz, dispone de un botón con el que se graba el audio. Este audio se manda al asistente virtual.
2. El asistente devuelve los datos que se solicitaron para mostrarlos en formato "***2D Picker".*** Gracias a este formato se podrá acceder a los diferentes elementos deslizando el dedo de arriba abajo.
3. Para acceder a los atributos de un elemento, basta con deslizar el dedo de izquierda a derecha. La acción se podrá acabar pulsando el botón mapa. Al pulsar este botón, se sincronizan móvil y telefono, lanzando el GPS que nos guiará hasta la ubicación del elemento.

Actualmente todos los dispositivos *smartwatch* con sistema *AndroidWea*r tienen el asistente virtual de Google integrado. Sin embargo, esto no implica que no se puedan añadir otros asistentes virtuales. Cabe destacar la diferencia entre el enfoque que posee el asistente virtual de Google (en smartwatch) con el que se ha realizado. El primero está enfocado a interaccionar con otras aplicaciones, mientras que el que se ha creado, está enfocado a la búsqueda de información desde el propio *wearable*. Por ello, la unión de ambos convierte a los *wearables* en una herramienta mucho más funcional y con una nueva potencia hasta entonces carente en estos dispositivos.

6.2 Desarrollo del módulo *wearable* y comunicación de este con la aplicación

6.2.1 Módulo *wearable*

MainActivity

Es la clase principal y contiene los siguientes métodos:

- ***OnCreate*:** Primer método al que se accede al crearse la clase.
  - Se asigna a esta clase una vista "***layout***" y se inicializan los elementos que la componen.
  - Se inicializa el ***ApiClient*** el cual notifica si se está conectado y en caso de error de conexión proporciona la información relativa a este.

- ***DisplaySpeechRecognizer*:** Inicia el reconocimiento de voz de Google. Se accede a él al pulsar el botón grabar del *wearable*.

- ***EnviarMensaje*:** Obtiene todos los nodos a los que está conectado (en este caso solo puede ser uno ya que se trata de un *wearable*), envía el mensaje a todos y mediante un *callback* en el caso de que el mensaje se haya recibido correctamente devolverá un *Success*. La conexión se realizará con el nodo que devuelva el *Success*.
- ***OnMessageReceived:*** Se extrae el path del mensaje recibido. En el caso de que se trate de *WEAR_ENVIAR_ITEMS_CARMEN* se extraen los datos y se envían a la clase ***PickerActivity***.

PickerActivity

Esta actividad cuando se la llama saca los extras del Intent para llamar posteriormente al método ***setUpData***. De esta forma se crean objetos ***QuoteList***, que se introducen en un array. Una vez que el array está formado llama al método ***setupGridViewPager*** donde se crea un elemento ***GridViewPager*** asociándole un adaptador al que se le pasa el array de objetos ***QuoteList***.

QuoteList

Se trata de una clase anónima que está contenida en la clase ***PickerActivity***. Cada objeto de esta clase contiene 3 campos: nombre, foto y ***quotes***. Estos objetos pueden usar los métodos de la clase: ***getTitle***, ***getText*** (el cual te devuelve el contenido de una de las posiciones del listado de ***quotes***), ***getPageCount*** (indica el número de tarjetas en dirección horizontal que se deben crean por cada elemento) y ***getImageResource*** (proporciona la imagen del elemento).

QuoteGridPagerAdapter

Es el encargado de visualizar todos los datos, de crear tantas tarjetas horizontales por elemento como indique el método ***getPageCount*** y de insertarlas sobre el fondo que indique el método ***getImageResource.*** Ambos métodos, como ya hemos visto son métodos contenidos en los objetos ***QuoteList***. Cuando llegue al último elemento en vez de crear un ***fragmentCard*** como los anteriores se crea un fragmento de la clase ***Map***.

Map

Esta clase extiende de **Fragment**. Carga una vista que contiene un botón con un icono de mapa que en el caso de ser pulsado abre la aplicación de mapas del *wearable* con las coordenadas que han sido pasadas mediante el método ***build*** de la clase anónima ***Builder***. El mapa se abre con la figuración de navegación modo *walking*; de esta forma guiará al usuario hasta la ubicación sin necesidad de sacar el móvil del bolsillo.

Builder

Clase anónima contenida dentro de la clase ***Map***. Contiene los siguientes métodos:

- ***Constructor:*** Recibe como parámetros las coordenadas.
- ***Build:*** Crea un fragmento ***Map*** y le pasa como argumentos las coordenadas.

6.2.2 Módulo Móvil

Dentro del método ***onCreate*** de la clase ***MainActivity*** perteneciente a la app del móvil se crea un objeto ***GoogleApiClient*** que, al igual que en el módulo *wearable* de la app, notifica cuando se está conectado e indica cuando se ha producido un error de conexión.

Además, se sobrescriben los siguientes métodos:

- ***onStart***, conectando el objeto ***GoogleApiClient***.
- ***onConnected,*** creando los *listeners* destinados a escuchar al *wearable* cuando nos envié un mensaje.
- ***onStop,*** eliminando los *listeners* creados en el ***onConnected*** y desconectando el objeto ***GoogleApiClient***.
- ***onMessageReceived***, creando un ***ApiConnectionListener*** al que se envía el texto proveniente del reconocedor de voz del *wearable*. En el caso de tener éxito se obtiene la respuesta proveniente del asistente virtual en la nube. Si esta respuesta contiene un campo de datos distinto de *null*, se obtienen los campos de los datos de interés (nombre, dirección, coordx, coordy y foto). Posteriormente se crean objetos (mediante los campos de interés) y con estos, un JsonArray que se envía al *wearable*.

7 Integración, pruebas y resultados [4]

Según Mike Cohn [12] "Tests should be automated. You know it. I know it. Agile methods insist on it. Yet, all too often we don't do enough of it, don't do it soon enough, or worse don't do it at all. I believe one big reason why we fall short is that we tend to automate at the wrong level. Most teams focus all their energy on unit testing and UI testing, while ignoring service-level testing altogether". Por esta razón Mike Cohn diseño la siguiente pirámide:

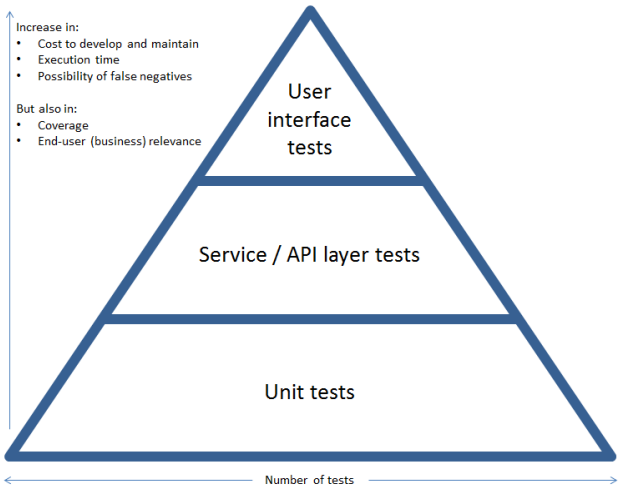


Figura 19: Pirámide de Mike Cohn

La realización de las pruebas se hizo siguiendo las recomendaciones de Mike Cohn, de esta forma la mayor cantidad de pruebas que se realizaron fueron unitarias, seguidas de las de integración y por último las pruebas de usuario final (estas últimas en menor cantidad).

Pruebas Unitarias (Unit tests)

Consisten en la ejecución de actividades que permitan verificar al desarrollador que los componentes unitarios están codificados bajo condiciones de robustez, esto es, soportando el ingreso de datos erróneos o inesperados. Así se demuestra la capacidad de tratar errores de manera controlada. Estas pruebas han de realizarse, y se realizaron, en los diferentes módulos del programa. Entre los grupos de pruebas unitarias usadas en la validación de nuestro código destacan:

- Librería **Va_sdk**: Esta librería es de vital importancia ya que deberá poder ser usada por cualquier persona. En ella se comprobó el correcto funcionamiento de todos los métodos que esta contiene. Cabe destacar tanto la comprobación en la extracción de los diferentes datos que son devueltos por el asistente virtual como aquellos métodos relacionados con la correcta comunicación con la nube de Google para el reconocimiento y síntesis de voz.
- Funciones relacionadas con la correcta extracción de datos a partir del fichero de configuración.

Pruebas de Integración (API layer tests)

Conjunto de pruebas que consisten en la comprobación de qué elementos del software, que interactúan entre sí, funcionan correctamente. Las principales pruebas de integración realizadas son:

- Correcta integración de la nube de Google (a la hora de recoger la petición del usuario) con la nube de Vocalia (a la hora de pasarle la petición transformada a texto).
- Integración con el *smartwatch* y funciones enfocadas en la comunicación con este.
- Pruebas relacionadas con el correcto flujo y creación de fragmentos.
- Correcta integración de los datos recopilados del fichero de configuración con nuestra app.

Pruebas de Aceptación / de Cliente

Estas pruebas son de vital importancia, en ellas se comprueba el correcto funcionamiento de la aplicación como un conjunto. Fueron realizadas por un grupo de personas externas al proyecto, de esta forma al no tener un conjunto prefijado de parámetros de entrada nos cercioramos de la solidez de nuestra aplicación.

8 Conclusiones y trabajo futuro

8.1 Conclusiones

Al término de este TFG se obtuvo un conjunto de resultados bastante satisfactorios. Por un lado, se dio un salto hacia el futuro creando una aplicación "Gateway". La cual, como bien indica la palabra, permitirá a aplicaciones externas ser usadas o usar asistentes de voz, comunicarse con la nube, etc. El futuro que se avecina se caracteriza por la gran cantidad de información que se dispondrá. Toda esta información deberá ser presentada de forma atractiva para el usuario y para ello surgirá la necesidad de crear asistentes virtuales capacitados para interaccionar tanto con información contenida en la nube como con los diferentes dispositivos que la proveen o la muestran. Esta aplicación, asienta las bases a la hora de incorporar asistentes virtuales en aplicaciones y en cómo comunicarnos con ellos, ya sea para mostrar datos, para interaccionar con aplicaciones de terceros o para conectarnos con otros dispositivos. En definitiva, proporciona un conjunto de posibles soluciones a la hora de realizar cualquier aplicación que use asistentes virtuales, independientemente de la funcionalidad con la que estos estén enfocados.

Por otra parte, se decidió dar un paso más allá, permitiendo que no sea solo el dispositivo móvil el que lleve todo el peso de la interfaz de usuario. Para ello, propusimos una nueva forma de realizar aplicaciones en dispositivos *wearable*, gracias a la cual, no se necesita el uso de notificaciones para el desempeño de diferentes acciones en estos dispositivos. Con esto les proveemos de una mayor independencia hasta entonces inexistente en la mayoría de ellos. Lo cual puede ser fundamental a la hora de mejorar la opinión de los usuarios sobre estos productos. Cosa que, en definitiva es posible que afecte en gran medida al mercado de los *smartwatches* pudiendo frenar el número de devoluciones que sufren actualmente (en torno al 30 %). Parámetro fundamental para incentivar aún más un mercado que está en auge.

8.2 Trabajo futuro

Este TFG admite diversas vías de mejora, desde el apartado visual hasta el de modularización:

- En el fichero de configuración, sustituir las FAQ por una url, de esta forma no solo se podrían ver las FAQ en formato página web sino que, también para el caso de preguntas asociadas a conversación sin la finalidad de obtener algún resultado, se podría mostrar en el fragmento superior un avatar[6] del asistente virtual.

- Aumentar la modularidad de la aplicación para *wearables*. Para realizarlo habría que pedir, al igual que hace la aplicación para móvil, un fichero de configuración y con él autoconfigurar la aplicación siguiendo la imagen de empresa. Esta configuración debería poderse realizar tanto desde el terminal móvil como desde el *wearable*.

[6] Avatar: Imagen 3D en movimiento que representa al asistente

- Añadir acceso a otras aplicaciones del *smartphone* desde la aplicación realizada como, por ejemplo, el Calendario.

Referencias

[1] Pedro Jiménez Martín, Jesús Sánchez Allende, "De Eliza a Siri: La Evolución", Revista de Ciencia, Tecnología y Medio Ambiente, Volumen XIII, 30, 2015.

[2] Oliver Miller, "A Conversation with ELIZA, The Electronic Therapist", THOUGHT CATALOG, 1 de Agosto de 2012, http://thoughtcatalog.com/oliver-miller/2012/08/a-conversation-with-eliza/ .

[3] "Cool, Sense", jabberwacky.com, 2009, http://www.jabberwacky.com/j2convbydate-QK14667 .

[4] Javier Zapata, "Niveles de Prueba del Software," Wordpress, 21 de enero de 2013, https://pruebasdelsoftware.wordpress.com/ .

[5] "Soluciones para internet of things,", Artificial Solutions, http://www.artificial-solutions.es/soluciones-nli/interfaz-de-lenguaje-natural/soluciones-para-internet-things/ .

[6] Ignacio Santiago, "Guerra de asistentes de voz: Cortana vs Siri vs Google Now", Ignacio Santiago, http://ignaciosantiago.com/asistentes-voz-windows-cortana-apple-siri-google-now-android/ .

[7] Koen Kas. "Besides smart glasses and smart watches, what are other type of wearable technology are the most likely to be adopted?", Quora, https://www.quora.com/Besides-smart-glasses-and-smart-watches-what-are-other-types-of-wearable-technology-are-the-most-likely-to-be-adopted#!n=12 .

[8] "Samsung no se equivoca: los wearables deben de ser independientes de nuestros smartphones", hipertextual 29 de agosto de 2014 http://hipertextual.com/archivo/2014/08/conectividad-3g-wearables/ .

[9] James Revelo, "Realizar Peticiones Http con la librería Volley en Android", 22 de febrero de 2015, http://www.hermosaprogramacion.com/2015/02/android-volley-peticiones-http/ .

[10] Dave Evans, "Internet of Things, la próxima evolución de Internet lo está cambiando todo", CISCO, 11, 2011.

[11] A.F van Woudenberg, "A Chatbot Manager, Chatbots and Dialogue Systems: A Hybrid Approach", Master Education in Computer Science, T76318, Open University of the Netherlands, Faculty of Management, Science and Technology, 90, 17 de junio del 2014.

[12] Mike Cohn, Test Automation: Let Service Be Your Middle Man, Scrum Alliance, 2 de diciembre de 2014, https://www.scrumalliance.org/community/spotlight/mike-cohn/december-2014/test-automation-let-service-be-your-middle-man .